University of Technology Sydney

Doctor of Philosophy

Algorithms for scheduling without preemptions

---

by
Julia Memar

Thesis supervisors: A/Prof Dr Yakov Zinder, Dr Hanyu Gu

February 2022

School of Mathematical and Physical Sciences

Declaration

I, Julia Memar, declare that this thesis, is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the School of Mathematical and Physical Sciences, Faculty of Science at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Signed: Production Note:
Signature removed prior to publication.

Date: 10 February 2022

Algorithms for scheduling without preemptions

by

Julia Memar

Submitted to the School of Mathematical and Physical Sciences
on February 10, 2022, in fulfillment of the
requirements for the degree of
Doctor of Phylosophy

Abstract

This thesis is concerned with algorithms for scheduling without preemptions and it contributes to research as follows.

The new area of research, which has gained attention only in the last 15 years, is concerned with flow shop models where the storage requirement varies from job to job and a job occupies the storage continuously from the start of its first operation till the completion of its last operation. This thesis contributes to research by developing a new approach of constructing feasible solutions for such flow shop problems with job-dependent storage. This approach utilises Lagrangian relaxation and decomposition - the techniques that have never been used before for such flow shop problems. In this thesis, several Lagrangian relaxation and decomposition-based heuristics are developed for NP-hard flow-shop problems with job-dependent storage and the effectiveness of these heuristics is demonstrated by the results of computational experiments.

In this thesis, a new discrete optimisation procedure is introduced. This optimisation procedure can be viewed as an alternative exact method to a branch and bound algorithm for a class of discrete optimisation problems with certain properties. This class includes several NP-hard scheduling problems. This discrete optimisation procedure is an iterative algorithm, that searches for a feasible solution with the objective value of the current lower bound or determines that such a solution does not exist. Various methods of how this search can be carried out are investigated, and these methods are compared computationally in application to a scheduling problem.

The worst-case analysis of a polynomial-time approximation algorithm for a maximum lateness scheduling problem with parallel identical machines, arbitrary processing times and arbitrary precedence constraints is provided. The algorithm is a modification of the Brucker-Garey-Johnson algorithm originally developed as an exact algorithm for the case of the problem with unit execution time tasks and precedence constraints represented by an in-tree. For the case when the largest processing time does not exceed the number of machines, a worst-case performance guarantee which is tight for arbitrary large instances of the considered maximum lateness problem has been obtained. It is shown that, if the largest processing time is greater than the num-

ber of machines, then the worst-case performance guarantee for the list algorithm, obtained by Hall and Shmoys, is tight.

Thesis supervisors: Associate Professor Dr Yakov Zinder, Dr Hanyu Gu

Acknowledgments

- I would like to express my gratitude to my supervisors, Associate Professor Dr. Yakov Zinder and Dr. Hanyu Gu, for their guidance, for countless enlightening talks and hundreds of hours spent to share with me their wisdom.

- I would like to thank my esteemed co-authors, Professor Dr. Alexander Kononov, Sobolev Institute of Mathematics, Siberian Branch of the Russian Academy of Sciences, Novosibirsk, Russia, and Dr. Gaurav Singh, Research and Innovation Manager, BHP, Perth, for sharing their knowledge and expertise with me.

- A huge thank you to my family - for their eternal patience and bearing with me and this thesis for all these years.

- Finally, I would like to dedicate this thesis to my mother, Lubov Kuzminichna Bazanova, who has always believed in me.

This doctoral thesis has been examined by a Committee of the School of Mathematical and Physical Sciences, Faculty of Sciences, UTS as follows:

Contents

List of Figures

List of Tables

Chapter 1

Introduction and motivation for research

This thesis is concerned with algorithms for scheduling without preemptions. Scheduling theory as a branch of operations research is dedicated to the optimal allocation of limited resources, over time, to a set of activities [6, 72, 87]. For more than sixty years scheduling algorithms have been used as a tool to improve efficiency in manufacturing, infrastructure, supply chains and computer systems - virtually in any facet of modern life. In the words of the excellent review on fifty years of research in scheduling [88], "scheduling has become a major field within operational research with several hundred publications appearing each year".

Many scheduling problems are exceptionally hard to solve: these scheduling problems are NP-hard. NP-hardness of a problem implies that the corresponding decision problem is NP-complete [6]. Further, if there is a polynomial-time algorithm to solve an NP-complete problem then any problem from the class NP can be solved in polynomial time, and hence the class of P problems equals to the class of NP problems. However, whether or not $P = NP$ is one of the millennium problems [15].

There is a wide variety of scheduling problems that are classified in terms of machine environment, job characteristics and optimality criteria. A change of one of the parameters of a problem, which may seem insignificant, can change the problem's complexity from being polynomially solvable to an NP-hard problem.

For example, consider the following classical scheduling problem: a set of tasks is processed on identical parallel machines, there are precedence constraints, all tasks are of unit execution time. The objective is to minimise the makespan - the maximum completion time among all tasks.

- If there are two parallel machines, then there are several polynomial algorithms to solve the problem [12, 27, 111];
- If there are three parallel machines, the question about the complexity of the problem is still open [30];
- If the number of the parallel machines is arbitrary, then the problem is $NP-$hard in a strong sense [71, 101].

Another classical scheduling problem is a flow shop problem: there is a set of jobs, and each job has the same number of operations. Each operation is processed on a particular machine corresponding to a stage of the flow shop, and the order of operations is the same for each job; the objective is to minimise the maximum completion time of all jobs:

- if there are two stages, then the problem is polynomially solvable [57];
- if there are more than two stages, the problem is NP-hard [31].

Flow shops with job-dependent storage requirement

The flow shop problem above, as well as many other classical flow shop problems assume that there is an infinite buffer between the stages, that is once a job has completed its processing on a current stage, and if the next stage is still occupied, this job does not prevent the current machine from processing another job [19, 87]. However, real-life applications may require more accurate models, such as flow shops with storage (a buffer)[69, 103]. The flow shop models with job-dependent storage requirement have come into the focus of research only in the last ten-fifteen years. In these models, the storage requirement varies from job to job and a job occupies the storage continuously from the start of its first operation till the completion of its last

operation. This thesis contributes to this research by developing a new approach of constructing feasible solutions for such flow shop problems with job-dependent storage. This approach utilises Lagrangian relaxation and decomposition - the techniques that never has been used before for such flow shop problems. In this thesis, several Lagrangian relaxation and decomposition-based heuristics are developed for NP-hard flow-shop problems with job-dependent storage and the effectiveness of these heuristics is demonstrated by the results of computational experiments.

Discrete optimisation procedure

Besides heuristics, which do not necessarily provide an optimal solution, another approach to deal with the NP-hard problems such as the aforementioned scheduling problems on parallel machines and flow shops, is to use exact algorithms. An exact algorithm is essentially an intelligent enumeration of all possible solutions. There is a wealth of literature on the branch and bound algorithm, the exact algorithm, first proposed in 1960 [65]. However, [112, 113] have described another exact method for solution of NP-hard problems on parallel machines. In these papers, it has been shown that this method is superior to the branch and bound algorithm, hence paving the ground for generalisation. In this thesis, a new discrete optimisation procedure is introduced. This optimisation procedure can be viewed as an alternative exact method for a class of discrete optimisation problems with certain properties. These properties are not too restrictive as they allow to include in this class several NP-hard scheduling problems. This discrete optimisation procedure is an iterative algorithm, that searches for a feasible solution with the objective value of the current lower bound or determines that such solution does not exists. Various methods of how this search can be carried out are investigated, and these methods are compared computationally in application to a scheduling problem.

Worst-case analysis of an approximation algorithm

Finally, this thesis provides an analysis of an approximation algorithm. Approximation algorithms are polynomial-time scheduling algorithms that generate feasible solutions within a guaranteed deviation from an optimal solution. Optimal solutions

are usually not known for NP-hard problems, thus making the task of obtaining a worst-case performance guarantee difficult [88]. The maximum lateness scheduling problem with parallel identical machines is a classical scheduling problem [6, 72, 87]). A significant attention in the literature was dedicated to the case of the problem when all tasks have unit execution times. Much less is known about the case of the problem when all jobs have arbitrary processing times and there are arbitrary precedence constraints - despite this case's role in the theory and practice (see, for example, [6, 72, 87]). Further, all known worst-case performance guarantee results, with the only exception of [46], were obtained for the criterion of maximum completion time - the particular case of the criterion of maximum lateness. In this thesis, this gap in the literature is addressed: a worst-case analysis of a polynomial-time approximation algorithm for the maximum lateness scheduling problem with parallel identical machines, with arbitrary processing times and arbitrary precedence constraints, is provided. The algorithm is a modification of the Brucker-Garey-Johnson algorithm [7], that was originally developed as an exact algorithm for the case of the problem with unit execution time tasks and precedence constraints represented by an in-tree. A worst-case performance guarantee is obtained for the case when the largest processing time does not exceed the number of machines. The guarantee is tight for arbitrary large instances of the considered maximum lateness problem. It is shown that, if the largest processing time is greater than the number of machines, then the worst-case performance guarantee for the list algorithm, obtained in [46], is tight.

All scheduling problems, considered in this thesis do not allow preemptions, meaning that once the processing of a job has been commenced, it should be completed without interruptions.

1.1 Thesis Organization

This thesis is organised as follows:

- *Chapter 2*: This chapter presents the background and literature overview for the problems and algorithms discussed in this thesis.

- *Chapter 3*: In this chapter a Lagrangian relaxation and decomposition-based approach to construct solutions for several NP-hard flow shop problems with a job-dependent buffer is presented. For each problem, a Largangian relaxation and decomposition-based heuristic is discussed, complemented by the results of computational experiments.

- *Chapter 4*: This chapter is dedicated to a discrete optimisation procedure for a class of discrete optimisation problems which is defined by certain properties of the boundary of the feasible region and level sets of the objective function. The considered solution method is an iterative procedure which at each iteration computes a lower bound on the optimal objective value and searches for a feasible solution attaining this bound. The chapter describes three search methods - descending search, ascending search, and their combination. These methods are illustrated by an application to a scheduling problem. The resultant algorithms are compared using computational experiments.

- *Chapter 5*: This chapter presents a worst-case analysis of a polynomial-time approximation algorithm for a maximum lateness scheduling problem with parallel identical machines, arbitrary processing times and arbitrary precedence constraints. The algorithm is a modification of the Brucker-Garey-Johnson algorithm. For the case when the largest processing time does not exceed the number of machines, a worst-case performance guarantee is obtained. This guarantee is tight for arbitrary large instances of the considered maximum lateness problem. It is shown that, if the largest processing time is greater than the number of machines, then the worst-case performance guarantee for the list algorithm, obtained in [46], is tight.

- *Chapter 6*: This final chapter gives a summary of the thesis and its contribution to research.

Chapter 2

Background and literature overview

The work in this thesis draws upon the following areas of the scheduling theory: flow shop problems with a buffer, Lagrangian relaxation methods applied to such flow shops; scheduling problems on parallel machines - approximate and exact algorithms. In this chapter, we will touch upon scheduling problems classification and provide a literature overview of these research areas.

2.1 Scheduling theory: classification of problems

Scheduling theory deals with an allocation of limited resources over time. The source of the first scheduling models were real-life processes, hence the terms in which the scheduling problems are described have inherited some terminology from the manufacturing/transportation problems [16]. Scheduling models are defined in terms of a set of jobs (or tasks) with certain characteristics, machine environment - what sort of machines are used to process the jobs and how many machines are used, and an objective function. A schedule is an allocation of time on the machines to each of the jobs. To construct a feasible schedule is to assign to each job a completion time in this schedule in a way that all constraints of the problem are satisfied. The following three-field notation is used to describe a scheduling problem: $\alpha|\beta|\gamma$ [6, 87], where α

signifies the machine environment, β defines jobs' characteristics and γ sets out the objective function. To describe the problems discussed in this thesis we will specify some values of these fields. A comprehensive classification of scheduling problems can be viewed, for example, in [6].

α Machine environment is the set of the machines $\{M_1, M_2, ..., M_q\}$, where q is a positive integer. The machine environments differ by how the machines work and process the jobs. For example, in parallel machines environment any job can be processed by any machine, and all machines are identical - that is each job requires the same processing time on any machine. Parallel machines are signified by $\alpha = P$, and if the number of machines is specified, for example, if there are two or three parallel machines, then these models are signified by $\alpha = P2$ and $\alpha = P3$, correspondingly. Another machine environment, which is considered in this thesis, is a flow shop. In this environment, each job is processed in stages, and each machine represents a stage. Each job is represented by the same number of operations, which have to be processed in the same order, and each operation is processed by one machine corresponding to this operation stage. Similarly, the number of stages in a flow shop can be specified, for example, $\alpha = F2$ signifies that this is a two-stage flow shop problem.

β A set of jobs $N = \{1, 2, ..., n\}$, where n is a positive integer, may have characteristics that are related to the jobs' properties, such as processing time or precedence constraints, or characteristics which specify how the jobs are processed. If a set of jobs has precedence constraints, then *prec* is included in the field β. Sometimes the form of the precedence constraints is specified, for example, if the precedence constraints are in the form of an intree, then *intree* is included. The field β may include the processing times of jobs. For example, $p_i = 1$ indicates that all jobs are of unit execution time (UET). If job's processing can be interrupted, then *prmt* is included in β, indicating that preemptions are allowed. If each job has a specific time, after which it can be processed and

not earlier, then r_i in the field β signifies that there are release times for each job. If there are communication delays, β would include the duration of the delays. For example, $c_{ij} = 1$ signifies that there are unit communication delays.

γ This field signifies an optimality criterion, that is the function for which we wish to find the optimal value on the set of all feasible schedules. In this thesis we will consider the problems which require finding the minimum value of a given criterion. Let $C_i(\sigma)$ be the completion time of job i in a schedule σ. The following optimality criteria are defined in terms of jobs' completion times:

- makespan, or $C_{max}(\sigma)$ - which is a maximum completion time of the given set of jobs and defined as $C_{max}(\sigma) = \max_{i \in N} C_i(\sigma)$;
- maximum lateness $L_{max}(\sigma)$ - each job i is assigned a due date d_i - the time when the job should be completed. Then the job's lateness is defined as $L_i(\sigma) = C_i(\sigma) - d_i$ and the $L_{max}(\sigma)$ is defined as $L_{max}(\sigma) = \max_{i \in N} L_i(\sigma)$;
- total completion time is defined as $\sum_{i \in N} C_i(\sigma)$ and total weighted completion time is defined as $\sum_{i \in N} w_i C_i(\sigma)$, where w_i is a non-negative weight assigned to a job $i \in N$;
- maximum tardiness $T_{max}(\sigma)$ - each job i is assigned a due date d_i - the time when the job should be completed. Then the job's tardiness is defined as $T_i(\sigma) = \max\{C_i(\sigma) - d_i, 0\}$ and the $T_{max}(\sigma)$ is defined as $T_{max}(\sigma) = \max_{i \in N} T_i(\sigma)$;
- total tardiness is defined as $\sum_{i \in N} T_i(\sigma)$ and total weighted tardiness is defined as $\sum_{i \in N} w_i T_i(\sigma)$, where w_i is a non-negative weight assigned to a job $i \in N$.

2.2 Flow shop problems and algorithms

Flow shop problems have been studied intensively the past few decades due to their theoretical difficulty and practical importance [87], however, flow shop problems are

notoriously difficult to solve, and they are known for their intractability [66]. One of the first results is concerned with $F2||C_{max}$ problem. This problem is polynomially solvable by Johnson's algorithm [57], but if the number of machines is greater than two, this problem is NP-hard [31]. Further, if a flow shop consists of two or three machines only, then for the C_{max} criterion it is sufficient to consider only permutation schedules, that is the schedules, where the order of jobs' operations on each machine is the same [14]. As the general flow shop problem even with three machines is NP-hard, a lot of efforts were put into developing effective heuristics for the permutation flow shop problem, where jobs are processed on each stage in the same order. The classical heuristics include Gupta-Palmer's heuristic [44, 85], which uses a certain priority list to construct the permutation; Campell-Dudek-Smith (CDS) heuristic [10], which uses Johnson's algorithm as an auxiliary algorithm to solve series of complementary two-stage flowshop problems; NEH heuristic proposed in [84] places the job with a longer total processing time first. The NEH heuristic is considered one of the most efficient algorithms for permutation flow shop problem [96]. A task of systematic classification and review of heuristics for the permutation flow shop problem with makespan criterion seems to be not an easy one, due to a large number of heuristics developed over the years and lack of common data to compare the performance. In the excellent review [90], the benchmark instances proposed in [97] were used to compare over twenty heuristics. Another review [23] aimed to classify heuristics according to the framework defined by index development, solution construction and solution improvement.

Other examples of NP-hard flow shop problems are [70]: the makespan problem with release times $F2|r_i|C_{max}$; the makespan problem with precedence constraints represented by a tree $F2|tree|C_{max}$; the maximum lateness problem $F2||L_{max}$. Some particular cases of the flow shop problems are polynomially solvable, such as the two machine flow shop makespan problem with precedence constraints represented by a tree and unit execution time jobs $F2|tree, p_i = 1|C_{max}$ [70], or the two-machine flow shop makespan problem with preemptions $F2|prmt|C_{max}$ [35]. However, the latter problem becomes NP-hard , if the number of machines is three - $F3|prmt|C_{max}$ [35].

The aforementioned problems are examples of a "classical" flow shop problem - the classical flow shop problem implicitly assumes that there is an infinite buffer between the stages, that is once a job has completed its processing on a current stage, and if the next stage is still occupied, this job does not prevent the current machine from processing another job [19, 87]. This assumption makes flow shop problems simpler - though many of them are still NP-hard even with this assumption, however, real-life applications may require more accurate flow shop models. These models include no-wait flow shops, flow shops with blocking and flow shops with buffers:

- **No-wait flow shops:** This model assumes that a job is processed through all stages of the flow shop without any wait between the stages. Such requirement can reflect the manufacturing processes, where no storage/buffer is available, or it is being dictated by the technology of the process, such as, for example, in steel manufacturing - steel has to be of a certain temperature while going through all technological stages; in food processing, canning of food should happen immediately after the food is prepared, or in various chemical processes, once a reaction has started on the first stage, it should be completed through all stages without delay to avoid deterioration of the product (see, for example, [19, 47, 91]). The two-stage no-wait flow shop makespan problem can be solved in polynomial time [34], however it has been shown in [86] that a no-wait flow shop makespan problem with 4 or more stages is an NP-hard problem. A few years later it was established in [89] that even the three-stage no-wait flow shop makespan problem is NP-hard.

- **Flow shops with blocking:** The model assumes that there is no storage/buffer to upload jobs once they have been processed by a machine, if the downstream machine is not available. Hence the job "blocks" the current machine till the next machine is free to process the job [19]. A comprehensive review of papers dedicated to the flow shops with blocking is provided in [82]. According to the review, the majority of papers are concerned with metaheuristics and local search algorithms.

- **Flow shops with intermediate buffers:** This model assumes that there is storage between stages where the job can be uploaded till the next stage machine becomes available and the buffer capacity is limited by some number of jobs. The model has been extensively studied in the literature:
  - it has been established in [86] that a makespan flow shop problem with intermediate buffers is NP-hard;
  - one of the early papers [69] formulates the flow shop problem with an intermediate buffer, describes three types of a buffer: limited (in a number of jobs), zero buffer, and an unlimited buffer; the paper also introduces heuristics for permutation and general flow shops with intermediate buffers, which are either an adaptation of heuristics for the flow shops with an unlimited buffer, or take the buffer constraint explicitly into account.
  - in [8] a tabu search algorithm for a makespan flow shop problem with intermediate buffers and an arbitrary number of stages is developed. The algorithm utilises a polynomial-time procedure that constructs a feasible schedule for the given sequence of jobs on each stage.
  - in [9], chapter "Job shop problems with limited buffers", formulations of job-shop and flow shop problems with various intermediate buffers are discussed: a general buffer problem, where any operation can be placed in a buffer; job-dependent buffer problem, where each job has its own buffer; pairwise buffer problem where a buffer depends on a pair of different stages' machines and an output and input buffer problems, where all jobs which either leave or about to be processed on a certain machine have to be placed in the buffer after or before being processed.
  - [3] and [76] are examples of the flow shop and job shop models with intermediate buffers inspired by real-world applications and develop heuristics that can provide sufficiently good solutions.
- **Flow shops with storage (buffer):** The models with the buffer requirement, which varies from job to job, and the buffer is occupied by a job for its entire

processing, have been studied only recently, though such models may better reflect the real-world applications [103]. The scheduling problems with such a buffer arise in supply chains when the change of vehicles involves unloading and loading, using certain storage space or in digital libraries, where files for presentations are downloaded from remote storage and stored in limited memory. In what follows we consider in detail the results on flow shops with storage (buffer), which varies from job to job, and the buffer is occupied by a job for its entire processing.

Flow shops with storage (buffer):

Perhaps [75] was the first publication in which a new flow shop model with a buffer requirement, which varies from job to job, and the buffer is occupied by a job for its entire processing, was formulated. This paper was motivated by the necessity to control a lag during a multimedia presentation, which requires a few files to be downloaded before the presentation can commence. The following problem was formulated:

$$F2|buffer, b(i)|C_{max}, \tag{2.2.1}$$

where $F2$ signifies that this is a two-machine flow shop problem, $buffer$ and $b(i)$ imply that there is a limited buffer capacity, which should not be violated at any time, and each jobs i occupies $b(i)$ units of the buffer from the start till the finish of the job's processing by the flow shop. It has been shown in [73] that (2.2.1) is NP-hard in a strong sense. The other early papers considered (2.2.1) with both "passive" and "active" pre-fetch - [60], where buffer space may be used more aggressively and allows a partial download of a job to resume broadcast and investigated the complexity of these "passive" and "active" models - [59]. The optimisation methods proposed in the early papers are branch-and-bound methods [73, 74], various implementations of variable neighbourhood search [59, 62]. All these publications are concerned with the minimisation of the time required for completion of all jobs or/and due dates based objective functions.

More recent papers contributed to the further research of complexity of flow shops

with the buffer, considered the objective functions other than C_{max}, proposed new models of flow shops with the considered buffer and applied the methods which have never been used before on such models:

- it has been proven in [25], that there are instances for which the set of all optimal schedules does not contain a permutation schedule, that is, a schedule in which both machines process the jobs in the same order. Even the decision problem, requiring an answer to the question of whether or not the given instance is one of the instances that do not have an optimal schedule that is a permutation one, is NP-complete;

- it has been shown in [38] the problem remains NP-hard in the strong sense even under the restriction that, on one of the machines, the jobs are to be processed in a given sequence;

- if the duration of each operation of a job does not exceed one-fifth of the buffer capacity, it has been proven in [61] that there exists a polynomial-time algorithm which constructs an optimal schedule;

- in [4] the (2.2.1) problem with variable buffer capacity $\Omega(t)$ was considered and it was established that the problem remains NP-hard in a strong sense even if all jobs' operations are of unit execution time;

- even in the restricted cases such as when for each job the duration of operation on a second-stage machine is the same and the buffer requirement is proportional to the duration of the job's operation on the first-stage machine the (2.2.1) remains NP-hard as shown in [68].

- in [20] a two-stage flexible flow shop has been considered, with first and second-stage machines put in disjoint pairs, each pair being assigned a buffer capacity, which varies from pair to pair. It has been shown that this problem is NP-hard for both objectives - makespan C_{max} and a total completion time $\sum C_i$.

- [108] gives a comprehensive analysis of complexity of the problem considered in this paper by providing a classification of the particular cases of the problem into families according to certain properties, and giving a complexity analysis of these cases.

2.3 Lagrangian Relaxation

Lagrangian relaxation is a technique that allows replacing a "difficult" problem with an easier one by "relaxing" some constraints or conditions. The Lagrangian relaxation was invented in 1797 by Joseph Louis Lagrange (1736 - 1813) as an extension of his work on equations for minimum/maximum functionals. Lagrangian relaxation is used extensively for solution of difficult combinatorial problems [33, 42, 43, 79]. In applications to integer programming problems, "bad" constraints are relaxed and placed to objective function with Lagrangian multipliers in a hope that the resulting problem with the remaining "easy" constraints can be solved for the current set of the multipliers. The resulting solution vector is used to update the Lagrangian multipliers.

In the early 70s in [50] and [51] Lagrangian relaxation was used to solve traveling salesman problem, and [32, 33] discussed some theoretical results for Lagrangian relaxation and the use of Lagrangian relaxation in linear programming-based branch-and-bound method. The classical paper on the application of Lagrangian relaxation to integer programming problems is [22], which is an excellent brief review and a guide to various aspects of Lagrangian relaxation application - basic concepts, construction and solution of Lagrangian relaxation problems, description of subgradient method, challenges of selection of the constraints to relax and construction of feasible solutions. This paper uses the results from [52], in which the convergence of the subgradient method was established. Another practical tutorial on the application of Lagrangian relaxation to integer programming problems is provided in [21].

[102] gives a comprehensive analysis of Lagrangian relaxation applications to scheduling problems, including integer programming formulations and Lagrangian

relaxation for single machine problems, identical and non-identical parallel machines and flow shops. Decomposition techniques such as surrogate relaxation are described, and it is demonstrated that Lagrangian relaxation provides tight lower bounds for branch-and-bound algorithms.

Further, Lagrangian relaxation can be adopted not only to provide lower bounds on an optimal solution but also to obtain feasible solutions and allow to evaluate the quality of these solutions. For example, [77] and [78] show that Lagrangian relaxation and decomposition applied to parallel identical machines provide both lower bounds and allow to develop algorithms that deliver nearly optimal feasible solutions. Another application of Lagrangian relaxation and decomposition, which allows deriving both lower bounds and a feasible solutions, is discussed in [18], where scheduling on parallel unrelated machines with additional resource problem is considered.

Lagrangian relaxation approach has been applied to flow shop problems such as, for example, hybrid flow shop problems modelling steel-making continuous casting process [79, 99]; there are only few papers on flow shop problems with buffers: for example, [56, 79, 98, 100] consider the implementation of Lagrangian relaxation-based algorithms in application to hybrid flow shop problems with intermediate buffers between the stages with the buffer capacity restricted by a number of jobs. However to the best of my knowledge, Lagrangian relaxation and decomposition techniques have never been applied to flow shops with a job-dependent buffer - this gap in literature is addressed in this thesis.

2.4 Scheduling problems on parallel machines

One of the most studied scheduling problems is the problem of minimising the maximum lateness on parallel machines, and its particular case, the so-called makespan problem, when all due dates are equal to zero. The interest in this problem arises, for example, from its practical importance related to the running of multiprocessor time-sharing computer systems. The important case related to the maximum lateness problem on parallel identical machines, which received significant attention in

the literature is the case where all tasks have unit execution times. Despite its role in the theory and practice (see, for example, [6, 72, 87]), much less is known about the case of the problem with arbitrary processing times and precedence constraints $P|\,prec\,|\,L_{max}$. The complexity of this problem is due to the presence of arbitrary precedence constraints, in particular even the $P|\,prec, p_j = 1\,|\,C_{max}$ problem is NP-hard in the strong sense [71, 101]. The $P|\,prec, p_j = 1\,|\,C_{max}$ problem remains NP-hard in the strong sense when the precedence constraints are in the form of a bipartite graph [110]. Furthermore, $P|\,prec, p_j = 1\,|\,L_{max}$ is NP-hard in the strong sense even if the precedence constraints are in the form of an out-tree [7]. Other factors which contribute to the complexity are arbitrary processing times and the restriction that a task's processing cannot be preempted, for example, $P\,||\,C_{max}$, which does not have precedence constraints, is NP-hard in the strong sense [29]. It has been shown in [95] that it is NP-hard to approximate $P|prec|C_{max}$ problem within any factor strictly less than two even in the case of unit processing times.

One of the main approaches used for the development of the approximation algorithms for scheduling problems is *list scheduling*. ***List algorithm*** is an iterative procedure, which at each iteration corresponds to some point in time t. The tasks form a priority list, and at the first iteration $t = 0$. At each iteration, the algorithm scans the list from the first position on the left to the right, and chooses the first task on the list, which is ready to be processed, and assigns the task a completion time $t + p_i$, where p_i is a processing time of task i. After that, the task is deleted from the list. The notion of the task's "readiness" depends on the problem at hand. For example, if precedence constraints and release times are present, the task is ready to be processed if t is greater than the task's release time and all task's predecessors if exist, have been processed. If the end of the list is reached, t is replaced by the next available time $t' > t$, and the next iteration starts. The procedure is repeated until the completion time is assigned for all tasks.

One of the first papers with an analysis of the performance of the list algorithm is provided in [36]. In this paper, the author applies the list algorithm to a makespan scheduling problem on parallel identical machines and investigates how the relaxation

of precedence constraints, changing the number of machines, decreasing the processing times of the tasks, or simply changing the priority list affects maximum completion time of a schedule. It has been shown that

$$\frac{C_{max}(s^L)}{C_{max}(s^*)} \leq 2 - \frac{1}{m}, \tag{2.4.1}$$

where $C_{max}(s^L)$ refers to the maximum completion time of the schedule s^L constructed by a list algorithm, and $C_{max}(s^*)$ is the maximum completion time of the optimal schedule s^*. The proof is based on the property of the list algorithm to assign a task to an idle machine if there is a task available, hence at least one task is processed in every time interval.

Problems with unit execution times

It easy is to see that for $P|prec, p_i = 1|C_{max}$ and $P|prec, p_i = 1|L_{max}$ scheduling problems there exists a list for an optimal schedule. Thus several two-phase approximation algorithms, which use the list algorithm, have been developed. The first phase of the algorithms is concerned with assigning priorities to the tasks and thus creating a priority list, and the second phase is constructing a schedule by the list algorithm according to the priority list, created in the first phase. The classical two-phase approximation algorithms are:

- **Hu's critical path algorithm:** In [55] $P|intree, p_i = 1|C_{max}$ scheduling problem is considered and it is shown that by assigning the length of the longest chain of successors as a priority to each task and executing tasks in non-increasing order of these priorities results in the optimal schedule.

- **Coffman and Graham algorithm:** [12] describes an algorithm for $P2|prec, p_i = p|C_{max}$ scheduling problem. The algorithm is an extension of Hu's algorithm, and it constructs an optimal schedule. However, it has been demonstrated that the algorithm does not necessarily provide an optimal schedule if the number of machines is arbitrary or tasks have arbitrary execution times.

- **Garey-Johnson algorithm:** this algorithm is presented in [27]. The al-

gorithm constructs an optimal schedule for $P2|prec, p_i = 1|L_{max}$ scheduling problem. Priority assignment is based on the following observations: tasks with smaller due dates should be processed sooner and each task has to be executed before all its successors. These two observations lead to the due date modification algorithm. Each task's modified due date d_i' is defined as

$$d_i' = \min\left\{d_i, \min_{j \in K(i)}\left\{d_j' - \left\lceil \frac{|\{k : k \in K(i) \text{ and } d_k' \leq d_j'\}|}{2} \right\rceil\right\}\right\},$$

where $K(i)$ is the set of all successors of task i and d_i is a task's due date before modification. The priority list is constructed in the non-decreasing order of the modified due dates. This algorithm was extended by the same authors to the $P2|prec, r_i, p_i = 1|L_{max}$ scheduling problem in [28].

- **Brucker-Garey-Johnson algorithm:** The proposed algorithm [7] generalises Hu's critical path algorithm, which was developed for the makespan problem, to the problem of minimising the maximum lateness: $P|intree, p_i = 1|L_{max}$. The priority of each task is its modified due date. The modification algorithm takes into account the modified due dates of the task's successors:

$$d_i' = \min\left\{d_i, \min_{j \in K(i)}\{d_j' - 1\}\right\}.$$

  The Brucker-Garey-Johnson algorithm solves $P|intree, p_i = 1|L_{max}$ in polynomial time.

Each of the above algorithms has been a subject of further research. Indeed, the worst-case performance of the Coffman-Graham algorithm was investigated in [64] followed by [5], where the following tight worst-case performance guarantee was obtained:

$$C_{max}(s^{CG}) \leq \left(2 - \frac{2}{m}\right) C_{max}(s^*) - r(m), \tag{2.4.2}$$

$$r(m) = \begin{cases} \frac{m-3}{m}, & \text{for odd } m\ ; \\ \frac{m-2}{m}, & \text{for even number of machines } m, \end{cases}$$

where m is the number of parallel identical machines, $C_{max}(s^{CG})$ refers to the maximum completion time of the schedule s^{CG} constructed by the Coffman-Graham algorithm, and $C_{max}(s^*)$ is the maximum completion time of the optimal schedule s^*. The 2.4.2 was further improved by [26], where it was proved that for $P|prec, p_i = 1|C_{max}$ problem with more than 3 parallel machines there exists an approximation polynomial algorithm with the following worst-case performance guarantee:

$$C_{max}(s^{GR}) \leq \left(2 - \frac{7}{3m+1}\right) C_{max}(s^*),$$

where $C_{max}(s^{GR})$ is the maximum completion time of the schedule s^{GR} constructed by the Gangal-Ranade algorithm, and $C_{max}(s^*)$ is the maximum completion time of the optimal schedule s^*. The Gangal-Ranade algorithm is not a list algorithm: it assigns each task a priority based on precedence constraints, however, the important feature of the algorithm is that it analyses predecessors of an unscheduled task and re-arranges already scheduled predecessors to allow the task to be scheduled earlier.

The worst-case performance guarantee for the Garey-Johnson algorithm for the maximum lateness problem with an arbitrary number of machines has been unknown for more than 30 years. In 2009 the tight worst-case performance guarantee was obtained for the algorithm generalised for an arbitrary number of machines for $P|prec, r_i, p_i = 1|L_{max}$ scheduling problem in [48]:

$$L_{max}(s^{GJ}) - L_{max}(s^*) \leq (1 - \alpha(m))(1 + \max_{1 \leq j \leq n} r_j),$$

where $L_{max}(s^{GJ})$ refers to the maximum lateness of the schedule s^{GJ} constructed by the Garey-Johnson algorithm, and $L_{max}(s^*)$ is the maximum lateness of the optimal schedule s^* and

$$\alpha(m) = \begin{cases} \frac{2}{m+1}, & \text{for odd } m \text{ ;} \\ \frac{2}{m}, & \text{for even } m. \end{cases}$$

The ratio derived for an even number of machines is the best known for this problem.

In [94] the Brucker-Garey-Johnson algorithm was generalised to the case of precedence constraints in a form of an arbitrary acyclic graph for the $P|prec, p_i = 1|L_{max}$

problem and the following tight upper bound for the worst-case performance was derived:

$$L_{max}(s^{BGJ}) \leq \left(2 - \frac{1}{m}\right) L_{max}(s^*) + \left(1 - \frac{1}{m}\right) \max_{1 \leq j \leq n} d_j - \left(\frac{m-1}{m}\right),$$

where $L_{max}(s^{BGJ})$ refers to the maximum lateness of the schedule s^{BGJ} constructed by the Brucker-Garey-Johnson algorithm, and $L_{max}(s^*)$ is the maximum lateness of the optimal schedule s^*. To obtain the bound, the structure of s^{BGJ} is investigated and a set of tasks, which can not occupy fewer time slots than in s^{BGJ}, is selected. This set includes a task with minimal completion time among all the tasks which provide the value of the objective function. The bound is obtained by estimating how much the value of the criterion on the set can be minimised in an optimal schedule.

The mentioned above proof technique was developed in [111], where it presents an approximation algorithm for $P|prec, p_i = 1|L_{max}$ scheduling problem and provides a tight worst-case performance guarantee. This is the best-known guarantee for the problem:

$$L_{max}(s^{ZR}) \leq \left(2 - \frac{2}{m}\right) L_{max}(s^*) + \left(1 - \frac{2}{m}\right) \max_{1 \leq j \leq n} d_j - r(m), \tag{2.4.3}$$

$$r(m) = \begin{cases} \frac{m-3}{m}, & \text{for odd } m \text{ ;} \\ \frac{m-2}{m}, & \text{for even number of machines } m, \end{cases}$$

where $L_{max}(s^{ZR})$ refers to the maximum lateness of the schedule s^{ZR} constructed by the Zinder-Roper algorithm and $L_{max}(s^*)$ is the maximum lateness of the optimal schedule s^*. The Zinder-Roper algorithm has two phases: it calculates tasks' priorities in the first phase and uses a list algorithm to construct a schedule in the second phase. Initially a priority of each task i is the difference of the maximum due date and the task's due date: $\gamma_i = \max_{1 \leq j \leq n} d_j - d_i$. It is shown that maximum lateness can be replaced by the following criterion:

$$G(s) = \max_{1 \leq j \leq n} (C_j + \gamma_j), \tag{2.4.4}$$

where C_j is the completion time of task j in a schedule s. Tasks' priorities are then

modified as follows. If a task has successors, a schedule s' is constructed with the help of the Zinder-Roper algorithm on the set of the task's successors. Then the modified priority of the considered task is set as the maximum between its initial priority and the value of the criterion $G(s')$. During the second phase of the algorithm, schedule s^{ZR} is constructed in non-increasing order of the modified priorities. As a corollary of (2.4.3), the algorithm constructs an optimal schedule in the case of two machines. The algorithm has been generalised to $P|prec, r_i, p_i = 1|L_{max}$ problem and shown that the bound (2.4.3) is tight in [106].

All the above-mentioned two-phase approximation algorithms can be associated with the family of *priority* algorithms. It has been shown in [107] that the Zinder-Roper algorithm is the strongest priority approximation algorithm for $P|prec, p_i = 1|L_{max}$ scheduling problem, thus providing the best possible worst-case performance guarantee.

Problems with arbitrary execution times

Possibly the earliest result concerning the worst-case performance of an approximate algorithm for makespan problem on parallel identical machines and the set of tasks scheduling with arbitrary processing times was provided in [37]. It was shown that the bound (2.4.1) can be improved for a set of independent tasks with arbitrary execution times, if the priority list is constructed according to LPT rule - "*longest processing time*" first. Then the worst-case performance has the following tight upper bound:

$$\frac{C_{max}(s^{LPT})}{C_{max}(s^*)} \leq \frac{4}{3} - \frac{1}{3m}, \tag{2.4.5}$$

where $C_{max}(s^{LPT})$ refers to the maximum completion time of the schedule s^{LPT} constructed by the list algorithm according to LPT rule, and $C_{max}(s^*)$ is the maximum completion time of the optimal schedule s^* and m is the number of parallel machines.

The *multifit* approximation algorithm, presented in [13], uses a technique, quite different from list scheduling. This technique allowed to improve (2.4.5) for $P||C_{max}$ scheduling problem. The algorithm is based on the observation that the problem of scheduling independent tasks on parallel machines and the bin-packing problem

are dual problems. Indeed, if the primal problem is the scheduling problem where a set of n independent tasks with arbitrary processing times p_j to be assigned to m parallel machines with the objective to minimise maximum completion time, then the dual problem would be to determine the maximum minimal capacity of the bins required to pack the n pieces with sizes p_j, such that the minimum number of the bins with this capacity does not exceed m. Thus the multifit algorithm employs a binary search between upper and lower bounds for the optimal value of C_{max}, and for each new capacity it uses an approximation algorithm to determine whether or not the optimal number of bins of this capacity necessary to pack all pieces, is exceeding m. The approximation bin-packing algorithm used on each iteration is *first fit decreasing* algorithm or FFD algorithm. The FFD algorithm re-arranges the pieces in non-increasing order of their sizes and places a piece in this order in the first bin it fits without violating the bin capacity. After k iterations the multifit approximation algorithm produces the following upper bound

$$\frac{C_{max}(s^{MF})}{C_{max}(s^*)} \leq 1.22 + \left(\frac{1}{2}\right)^k,$$

where $C_{max}(s^{MF})$ refers to the maximum completion time of the schedule s^{MF} constructed by the multifit algorithm, and $C_{max}(s^*)$ is the maximum completion time of the optimal schedule s^*. This bound was improved in [105]:

$$\frac{C_{max}(s^{MF})}{C_{max}(s^*)} \leq \frac{13}{11},$$

and the bound is tight. A similar approach was applied in [53], where a polynomial approximation scheme for $P||C_{max}$ scheduling problem was introduced. This scheme delivers a solution s for any $\varepsilon > 0$ such that

$$C_{max}(s) \leq (1+\varepsilon)C_{max}(s^*),$$

where s^* is an optimal schedule. Here we note that the algorithm is polynomial in $O((\frac{n}{\varepsilon})^{\frac{1}{\varepsilon^2}})$, thus making it infeasible for real-life applications. To give more practical

estimation of the algorithm performance, it is shown that after k iterations, the resulting solution s^k provides the following approximation:

$$C_{max}(s^k) \leq (1+\varepsilon)(1+2^{-k})C_{max}(s^*).$$

In [58] the $P|intree, p_i|C_{max}$ scheduling problem was considered and Hu's algorithm was applied to a set of tasks with arbitrary processing times and precedence constraints represented by an in-tree. Hu's idea of assigning a length of the path from a task to the root of a tree as a task's priority was generalised to the case of arbitrary processing times, and the tasks were executed in non-increasing order of priorities by a list algorithm: a priority or a level l_j of a task j is defined by both its processing time and priority of its successor i: $l_j = p_j + l_i$. Hu's critical or the longest path algorithm is known to be optimal for in-trees with tasks of unit execution time; it is shown that in the case of arbitrary processing times, the algorithm is "almost optimal" [58]:

$$C_{max}(s^{Hpmtn}) \leq C_{max}(s^*) \leq C_{max}(s^{LP}) \leq C_{max}(s^{Hpmtn}) + p_{max} - \frac{p_{max}}{m},$$

where s^{Hpmtn} - is the schedule obtained by Hu's longest path algorithm with preemptions allowed after each unit of time, s^* is the optimal schedule without preemptions and s^{LP} is the schedule obtained by the longest path (LP) algorithm without preemptions.

Graham's conjecture that the bound (2.4.1) for list algorithm can be improved in some cases was confirmed in [63]. If precedence constraints are represented by a tree and LP rule is used to compile the list for a list algorithm, then the following bound holds for $P|tree|C_{max}$ problem:

$$\frac{C_{max}(s^{LP})}{C_{max}(s^*)} \leq 2 - \frac{2}{m+1},$$

where $C_{max}(s^{LP})$ refers to the maximum completion time of the schedule s^{LP} constructed by the LP algorithm - list algorithm with the list comprised according to

the LP rule, and $C_{max}(s^*)$ is the maximum completion time of an optimal schedule s^*. The LP list is constructed in the non-decreasing order of the longest chain of a given task's predecessors, including the task's processing time.

A similar result is derived in [104], where the problem on parallel machines with delivery times is considered and the objective function is

$$H_{max}(s) = \max_{1 \leq i \leq n} \left(C_i(s) + q_i \right), \tag{2.4.6}$$

where $C_i(s)$ is the completion time of task i in a schedule s and q_i a delivery time of the task i. In the three-filed notation this pronlem could be described as $P|q_i|H_{max}$. It is shown that the criterion of minimising the maximum lateness is equivalent to minimizing the criterium (2.4.6). The proposed *Mixed Delivery and Processing Time List Scheduling* heuristic (MLS heuristic) is a list algorithm with the priorities which take into account both - the task's processing time and its delivery time: $w(i) = (m-1)p_i + mq_i$, where m is the number of parallel machines, p_i and q_i are processing and delivery times for task i, correspondingly. It is shown that the worst-case upper bound for the MLS heuristic is

$$\frac{H_{max}(s^{MLS})}{H_{max}(s^*)} \leq 2 - \frac{2}{m+1},$$

where s^{MLS} is the schedule constructed by the MLS heuristic, and s^* is an optimal schedule.

The *Naive* algorithm for $P|r_i|L_{max}$ problem is described in [45]. The naive algorithm is essentially a list algorithm, in which at any iteration only the tasks with a release time greater than the current time t are scheduled in the non-decreasing order of due dates; t is selected as the minimum of the largest of busy times among all machines. The following tight upper bound for the worst-case performance is derived:

$$L_{max}(s^N) - L_{max}(s^*) \leq \left(2 - \frac{1}{m}\right) p_{max}, \tag{2.4.7}$$

where $L_{max}(s^N)$ refers to the maximum lateness of the schedule s^N constructed by

the naive algorithm, $L_{max}(s^*)$ is the maximum lateness of the optimal schedule s^* and p_{max} is the maximum processing time of the considered set of tasks. In case of equal processing times the (2.4.7) can be improved:

$$L_{max}(s^N) - L_{max}(s^*) \leq p, \tag{2.4.8}$$

where p is the processing time of any task. Further, if all due dates and release times are multiples of p, the naive algorithm solves the scheduling problem in polynomial time.

The approach applied to $P|r_j|L_{max}$ scheduling problem in [46] results in an almost optimal schedule for the problem. The problem is presented in a "delivery" form with L_{max} replaced by the maximum completion time with delivery times (2.4.6), and it is assumed that all tasks can be delivered at the same time. The approach utilises the notions of outline and Optimal Outline Scheme (OOS). An outline is some partial information about a schedule, such that the schedule can be restored using its outline; Optimal Outline Scheme is the algorithm, which for any given outline of a schedule s constructs a schedule s' with the value of the objective function not greater than that of s. Using the definition, the $(1+\varepsilon)$OOS allows to find "close enough" optimal schedule if the number of possible outlines for the OOS is polynomial, or a polynomial-size subset of outlines which includes an optimal solution, can be determined. The solution is found by running OOS for each possible outline for the given instance and selecting the best (minimal) solution. The $(1+\varepsilon)$OOS guarantees that for any $\varepsilon > 0$ the objective value of the found solution will be only $1+\varepsilon$ of the optimal value:

$$H_{max}(s) \leq (1+\varepsilon)H_{max}(s^*),$$

where s^* is an optimal schedule. The algorithm considers the various combinations of effective release times - the modified release times, such that there exists a feasible schedule with the modified release times. For each combination, the algorithm splits the tasks into sets according to their processing times and effective release dates. The long tasks in each group are followed by the small tasks, scheduled in a greedy

manner, so the total processing time of the small tasks in each group just exceeds a certain constant, which in turn depends on the value of ε.

Another classical result discussed in [46] is the upper bound on the performance of list algorithm for $P|prec, r_j|L_{max}$ scheduling problem:

$$\frac{H_{max}(s^L)}{H_{max}(s^*)} < 2, \tag{2.4.9}$$

where s^L is a schedule constructed by a list algorithm and s^* is an optimal schedule.

Exact algorithms

As exact methods for solving NP-hard problems are concerned, the branch and bound method is a very popular exact algorithm [6, 72, 87], first proposed in 1960 in [65]. The branch and bound algorithm splits the decision space into sub-regions (branching), and each sub-region is the basis for another branching. There is a wealth of literature dedicated to various aspects of the method applied to scheduling on parallel machines, such as efficient lower bounds and pruning techniques: [1, 2, 11, 17, 67, 83] - to name a few.

An alternative exact method was proposed and implemented in [112, 113]: the considered solution method is an iterative procedure that at each iteration computes a lower bound on the optimal objective value and searches for a feasible solution attaining this bound. This method was implemented for the following problems: $P|prec,\ p_{ij} = 1|C_{max}$ - in [112], and for $P|prec,\ c_{ij},\ p_{ij} = 1|C_{max}$ - in [113], and the computational experiments have demonstrated the superiority of the method in comparison with a conventional branch-and-bound algorithm. Chapter 4 of this thesis extends the results of these two papers.

Chapter 3

Lagrangian relaxation and decomposition-based algorithms for flow shops with job-dependent buffer requirements

The results of this chapter have been published in the following conference proceedings and a journal:

- [38]: Hanyu Gu, Alexander Kononov, Julia Memar, and Yakov Zinder. "Efficient lagrangian heuristics for the two-stage flow shop with job dependent buffer requirements". Journal of Discrete Algorithms, 52-53: pp.143 – 155, 2018.
- [61]: Alexander Kononov, Julia Memar, and Yakov Zinder. "Flow shop with job–dependent buffer requirements—a polynomial–time algorithm and efficient heuristics". In International Conference on Mathematical Optimisation Theory and Operations Research, pp 342–357. Springer, 2019.
- [39]: Hanyu Gu, Julia Memar, and Yakov Zinder. "Scheduling batch processing in flexible flowshop with job dependent buffer requirements: Lagrangian relaxation approach". In International Workshop on Algorithms and Computation, pp 119–131. Springer, 2018.

- [41]: Hanyu Gu, Julia Memar, and Yakov Zinder. "Improved Lagrangian relaxation-based optimisation procedure for scheduling with storage". IFAC-PapersOnLine 52, no. 13, pp 100-105. 2019.

The results of this chapter have been presented at the following conferences:

- the 28th Workshop on Combinatorial Algorithms IWOCA 2017, Newcastle, Australia, 17-21 July 2017.
- the 12th International Conference, WALCOM 2018, Dhaka, Bangladesh, March 3-5, 2018.
- the International conference on Mathematical Optimisation Theory and Operations Research MOTOR 2019, Ekaterinburg, Russian Federation, 8-12 July 2019.
- the 9th IFAC Conference on Manufacturing, Management and Control MIM 2019, Berlin, Germany, 27-30 August 2019.

3.1 Introduction

In this chapter, a Lagrangian relaxation and decomposition-based approach is applied to NP-hard two-stage flow shop scheduling problems with a job-dependent buffer. This approach allowed to develop heuristics - the algorithms, which do not guarantee an optimal solution, however, these algorithms are efficient in practice. This approach can be summarised as follows:

1. an integer linear programming formulation for a considered problem is developed;

2. some "bad" constraints in this formulation are dualised and a Lagrangian relaxation is obtained;

3. this relaxation is then decomposed into subproblems which are solved separately by a recursive procedure, developed in this chapter;

4. to find a good set of Lagrangian multipliers for the Lagrangian relaxation, the standard iterative procedure known as the subgradient method [22] is utilised:

4.1 at each iteration the Lagrangian relaxation for the current set of Lagrangian multipliers is solved;

4.2 as a result of the Lagrangian relaxation permutations of jobs are obtained - one for each stage. These permutations do not necessarily represent a feasible schedule - the algorithms developed in this chapter are used to construct a feasible schedule;

4.3 the value of the objective function for this feasible schedule is compared with the best value of the objective function found so far, and the best value is used as the current upper bound on the optimal value of the objective function in the calculation of a new set of Lagrangian multipliers.

5. after a chosen number of iterations, the feasible schedule with the best value of the objective function found in the course of these iterations is considered as the solution produced by the Lagrangian heuristic.

All computational experiments for this chapter were conducted on a personal computer with Intel Core $i5$ processor $CPU@1.70Ghz$, using Ubuntu 14.04 LTS, with base memory 4096 MB. The algorithms were implemented in C programming language. The CPLEX_OPTIM_STUDIO_12.6.1 software was also used for the computational experiments.

Each of the considered in this chapter problems is NP-hard in a strong sense:

- it has been shown in [31], that a two-stage flow shop problem without a job-dependent buffer and batches, and the objective of the total completion time is strongly NP-hard - this problem is a particular case of the two considered problems: $F2|buffer|\Sigma w_i C_i$ and $F2|buffer, batch|\sum w_k T_k$;
- the strong NP-hardness of $F2|buffer|C_{max}$ problem is due to [73].

This chapter is organised as follows:

Section 3.2: Lagrangian relaxation and decomposition is applied to $F2|buffer|\Sigma w_i C_i$ problem; several ways to construct a feasible schedule are proposed; the resulting algorithms are compared computationally;

Section 3.3: Lagrangian relaxation and decomposition is applied to $F2|buffer|C_{max}$ problem; the resultant heuristic is compared computationally with two other algorithms;

Section 3.4: Lagrangian heuristic developed to solve $F|buffer, batch|\sum w_k T_k$ problem. Several ways to improve the performance of this heuristic are discussed, and the resultant algorithms are compared computationally.

3.2 Two-stage flow shop with storage and objective to minimise total weighted completion time

3.2.1 Problem Description

In what follows, the Lagrangian relaxation and decomposition are applied to the following problem. The set of n jobs $N = \{1, ..., n\}$ is to be processed by two machines - the first-stage machine and the second-stage machine. Each job i is processed on the first-stage machine during p_i^1 (the first operation of the job) and on the second-stage machine during p_i^2 (the second operation of the job) time units. All processing times are integer. The second operation of a job can commence on the second-stage machine only after the completion of its first operation on the first-stage machine. Once an operation has started, it cannot be interrupted, i.e. no preemptions are allowed. Each machine can process at most one job at a time, and each job can be processed by at most one machine at a time. The processing of jobs commences at time $t = 0$.

To be processed, each job i requires $b(i)$ units of the buffer space. This buffer space is occupied by a job continuously from the start of its first operation till the completion of its second operation. At any point in time t the buffer capacity Ω can not be exceeded by the total buffer requirement of all jobs that started their processing before or at t and have a completion time of their second operation greater than t. Similar to [59], [62], [73], [74] and [75] it is assumed that the buffer requirement of each job is determined by the duration of the first operation: $b(i) = p_i^1$ for all $i \in N$.

For each job i, let S_i^1 and S_i^2 be the starting times of the job's first and second operation, respectively. The goal is to construct a schedule with the smallest total weighted completion time $\sum_{i \in N} w_i C_i$, where w_i is a positive weight, characterising i, and $C_i = S_i^2 + p_i^2$ is the completion time of job i. The considered problem is NP-hard in a strong sense, as even a particular case of the problem, when all weights are equal to one and there is no buffer, is NP-hard in a strong sense [31].

3.2.2 Lagrangian relaxation

In any optimal schedule, the completion time of a job can not exceed the following value of the planning horizon T :

$$T = \sum_{i \in N} (p_i^1 + p_i^2).$$

For each $i \in N$, integer $0 \leq t < T$, $m \in \{1, 2\}$, let

$$x_{it}^m = \begin{cases} 1, & \text{if } S_i^m = t; \\ 0, & \text{otherwise.} \end{cases}$$

Each decision variable x_{it}^m signifies whether or not job i has started on machine m at time t. Similar to [62], the considered scheduling problem can be formulated as the following integer linear program:

$$\min \sum_{i=1}^{n} w_i \left(\sum_{t=1}^{T-1} t x_{it}^2 + p_i^2 \right) \tag{3.2.1}$$

subject to

$$\sum_{t=0}^{T-1} x_{it}^m = 1, \quad \text{for } 1 \leq i \leq n \text{ and } m \in \{1, 2\} \tag{3.2.2}$$

$$\sum_{i=1}^{n} \sum_{\tau=\max\{0, t-p_i^m+1\}}^{t} x_{i\tau}^m \leq 1, \quad \text{for } 0 \leq t < T \text{ and } m \in \{1, 2\} \tag{3.2.3}$$

$$\sum_{t=1}^{T-1} t x_{it}^2 - \sum_{t=1}^{T-1} t x_{it}^1 \geq p_i^1, \quad \text{for } 1 \leq i \leq n \tag{3.2.4}$$

$$\sum_{i=1}^{n} b(i) \left[\sum_{\tau=0}^{t} x_{i\tau}^1 - \sum_{\tau=0}^{t-p_i^2} x_{i\tau}^2 \right] \leq \Omega, \quad \text{for } 0 \leq t < T \tag{3.2.5}$$

$$x_{it}^m \in \{0, 1\}, \quad \text{for } 1 \leq i \leq n, \ 0 \leq t < T, \ \text{and } m \in \{1, 2\} \tag{3.2.6}$$

Constraint (3.2.2) ensures that each job is scheduled only once on each machine; constrains (3.2.3) and (3.2.5) are capacity constraints for the number of machines on each stage and the buffer capacity, correspondingly; (3.2.4) enforces the order of operations for each job and (3.2.6) indicates that all decision variables are binary.

Dualising (3.2.3) and (3.2.5) for chosen nonnegative Lagrange multipliers v_{tm} and u_t, where $0 \leq t < T$ and $m \in \{1, 2\}$, gives the following Lagrangian relaxation:

$$\min \sum_{i=1}^{n} w_i \left(\sum_{t=1}^{T-1} t x_{it}^2 + p_i^2 \right) + \sum_{t=0}^{T-1} \sum_{m=1}^{2} v_{tm} \left(\sum_{i=1}^{n} \sum_{\tau=\max\{0, t-p_i^m+1\}}^{t} x_{i\tau}^m - 1 \right)$$
$$+ \sum_{t=0}^{T-1} u_t \left(\sum_{i=1}^{n} b(i) \left[\sum_{\tau=0}^{t} x_{i\tau}^1 - \sum_{\tau=0}^{t-p_i^2} x_{i\tau}^2 \right] - \Omega \right)$$

subject to (3.2.2), (3.2.4) and (3.2.6). Let v be the set of all v_{tm} and u be the set of all u_t. Let $LR(v, u)$ be the optimal value of the objective function of the Lagrangian relaxation above. For each $i \in N$, let $Z_i(v, u)$ be the optimal value of the objective function of the integer linear program

$$\min w_i \sum_{t=1}^{T-1} t x_{it}^2 + \sum_{t=0}^{T-1} \sum_{m=1}^{2} v_{tm} \sum_{\tau=\max\{0, t-p_i^m+1\}}^{t} x_{i\tau}^m + b(i) \sum_{t=0}^{T-1} u_t \left(\sum_{\tau=0}^{t} x_{i\tau}^1 - \sum_{\tau=0}^{t-p_i^2} x_{i\tau}^2 \right) \tag{3.2.7}$$

subject to

$$\sum_{t=0}^{T-1} x_{it}^m = 1, \quad \text{for } m \in \{1, 2\} \tag{3.2.8}$$
$$\sum_{t=1}^{T-1} t x_{it}^2 - \sum_{t=1}^{T-1} t x_{it}^1 \geq p_i^1 \tag{3.2.9}$$
$$x_{it}^m \in \{0, 1\}, \quad \text{for } 0 \leq t < T \text{ and } m \in \{1, 2\} \tag{3.2.10}$$

It is easy to see that

$$LR(v, u) = \sum_{i=1}^{n} Z_i(v, u) + \sum_{i=1}^{n} w_i p_i^2 - \sum_{t=0}^{T-1} \sum_{m=1}^{2} v_{tm} - \Omega \sum_{t=0}^{T-1} u_t \tag{3.2.11}$$

Hence, for chosen Lagrange multipliers, the Lagrangian relaxation can be solved by solving n separate integer linear programs (3.2.7) - (3.2.10). Each of these n problems can be solved with the technique described below.

Recursive procedure

According to (3.2.8), for each $i \in N$ exactly one x_{it}^1 and exactly one x_{it}^2 must be equal to 1 and all others must be zero. If $x_{is}^1 = 1$ and $x_{ir}^2 = 1$ for some s and r , then the

corresponding value of the objective function (3.2.7) is

$$w_i r + \sum_{t=s}^{s+p_i^1-1} v_{t1} + \sum_{t=r}^{r+p_i^2-1} v_{t2} + b(i) \sum_{t=s}^{r+p_i^2-1} u_t.$$

Observe that, by virtue of (3.2.9), $s \leq r - p_i^1$. Consider function $f(r)$ defined for all $p_i^1 \leq r \leq T - p_i^2$ as follows

$$f(r) = \min_{0 \leq s \leq r-p_i^1} \left(\sum_{t=s}^{s+p_i^1-1} v_{t1} + b(i) \sum_{t=s}^{r+p_i^2-1} u_t \right). \tag{3.2.12}$$

If $r > p_i^1$, then $f(r)$ can be found as

$$f(r) = \min \left[f(r-1) + b(i) u_{r+p_i^2-1}, \sum_{t=r-p_i^1}^{r-1} v_{t1} + b(i) \sum_{t=r-p_i^1}^{r+p_i^2-1} u_t \right] \tag{3.2.13}$$

The initial value of $f(r)$ is for $r = p^{1_i}$:

$$f(p_i^1) = \sum_{t=0}^{p_i^1-1} v_{t1} + b(i) \sum_{t=0}^{p_i^1+p_i^2-1} u_t. \tag{3.2.14}$$

Hence, using (3.2.13) and (3.2.14), all values $f(r)$ can be calculated recursively in $O(T)$ operations, and by virtue of

$$Z_i(v,u) = \min_{p_i^1 \leq r \leq T-p_i^2} \left(f(r) + w_i r + \sum_{t=r}^{r+p_i^2-1} v_{t2} \right),$$

the integer linear program (3.2.7) - (3.2.10) can be solved in $O(T)$ operations.

3.2.3 Lagrangian heuristics

The Lagrangian relaxation specifies two orders: the order in which the jobs are to start on the first-stage machine and the order in which the jobs are to start on the second-stage machine. It is convenient to refer to these orders as permutations π_1 and π_2 of the set N. Please note that a feasible schedule where the jobs are processed on

the first-stage machine according to π_1 and on the second-stage machine according to π_2 may not exist. In this section, two algorithms that construct a feasible schedule are described. The Algorithm NO-WAIT is a greedy algorithm in which the permutations π_1 and π_2 set the priority between the jobs on each machine and the algorithm can violate the order specified by these permutations. Whereas the Algorithm WAIT strictly follows these permutations if possible. A necessary and sufficient condition of the existence of a feasible schedule for a pair of permutations is also presented.

Description of NO-WAIT algorithm

The NO-WAIT algorithm uses Stage 1 procedure to schedule jobs on the first-stage machine and Stage 2 procedure - to schedule jobs on the second-stage machine. Stage 1 selects the first unscheduled job in the permutation π_1, that "fits" in the buffer and assigns this job on the first-stage machine. Stage 1 goes through π_1 and continues to assign jobs to the first-stage machine while there are jobs which can be placed in the buffer or the last job in π_1 is reached. If no job is placed in the buffer during current iteration of Stage 1, the algorithm proceeds to Stage 2 procedure. Stage 2 assigns the first available job in the permutation π_2 to the second-stage machine. Stage 2 continues to assign jobs to the second-stage machine while there are unscheduled available jobs.

Denote by t_1 and t_2 the current starting time on the first and the second-stage machine, correspondingly. Let In be the sum of buffer requirements of the jobs which are currently in buffer. Denote by U_1 the set of unscheduled jobs, by U_2 the set of jobs assigned on the first-stage machine and not yet assigned to the second-stage machine, and by U_B the set of jobs in the buffer at time t. For convenience, a dummy job $n+1$ is added at the end of permutations π_1 and π_2 and set $b(n+1) = 0$. The NO-WAIT algorithm can be summarized as follows:

Algorithm NO-WAIT

1: **Set** $U_1 = N \cup \{n+1\}$, $U_2 = \{n+1\}$, $U_B = \emptyset$, $t_1 = 0$, $t_2 = 0$, $In = 0$, $S^1_{n+1} = 0$;

2: **Set** $p^1_{n+1} = 0$, $p^2_{n+1} = 0$, $k_1 = 0$, $k_2 = 0$, $k = 0$.

3: **while** $U_1 \neq \{n+1\}$(the stopping criterion is not fulfilled) **do**

4: Let $k = \min\{\pi_1^{-1}(j) | j \in U_1 \ \& \ In + b(j) \leq \Omega\}$. **Set** $i = \pi_1(k)$.

5: Stage 1

6: Stage 2

7: **end while**

*Stage*1 **procedure**

1: **while** $i < n + 1$ **or** $t_1 < t_2$ **do**

2: **if** $i < n + 1$ **then**

3: **set** $S_i^1 = t_1$, $U_1 = U_1 \setminus \{i\}$, $U_2 = U_2 \cup \{i\}$, $U_B = U_B \cup \{i\}$, $t_1 = t_1 + p_i^1$, $In = In + b(i)$;

4: **else**

5: Let $\tau = \min\{S_j^2 + p_j^2 | j \in U_B\}$. **Set** $t_1 = \tau$.

6: **end if**

7: **for** $j \in U_B$ **do**

8: **if** $S_j^2 + p_j^2 \leq t_1$ **then**

9: $U_B = U_B \setminus \{j\}$, $In = In - b(j)$;

10: **end if**

11: **end for**

12: Let $k_1 = \min\{v > k | \pi_1(v) = j \ \& j \in U_1 \ \& \ In + b(j) \leq \Omega\}$.

13: **Set** $i = \pi_1(k_1)$, $k = k_1$.

14: **end while**

*Stage*2 **procedure**

1: **Set** $k = 0$.

2: Let $\tau = \min\{S_j^1 + p_j^1 | j \in U_2\}$. **Set** $t_2 = \max\{t_2, \tau\}$.

3: **while** $k_2 \leq n$ **do**

4: Let $k_2 = \min\{v > k | \pi_2(v) = j \ \& j \in U_2 \ \& \ S_1^j + p_1^j \leq t_2\}$. **Set** $i = \pi_2(k_2)$, $k = k_2$.

5: **if** $i < n + 1$ **then**

6: **set** $S_i^2 = t_2$, $U_2 = U_2 \setminus \{i\}$, $t_2 = t_2 + p_i^2$.

7: **if** $S_i^2 + p_i^2 \leq t_1$ **then**

8: $U_B = U_B \setminus \{i\}$, $In = In - b(i)$.

9: **end if**

10: **end if**

11: **end while**

According to line 3 of the NO-WAIT algorithm and line 12 of the Stage 1 procedure the buffer capacity is not exceeded at any moment of time, since a job is scheduled on the first-stage machine only if there is sufficient space in the buffer to accommodate the job. At each iteration of the cycle **while** the Stage 2 procedure assigns to the second-stage machine only the jobs which have been already assigned by the Stage 1 procedure to the first-stage machine. Further, the Stage 1 procedure is called until there are unscheduled jobs. Thus the NO-WAIT algorithm constructs a feasible schedule.

Description of WAIT algorithm

First, a necessary and sufficient condition of the existence of a feasible schedule for a pair of permutations is presented. A pair of permutations (π_1, π_2) is *feasible* if there exists a schedule such that the jobs on the first-stage machine are processed in the order specified by π_1 and on the second-stage machine the jobs are processed in the order specified by π_2.

Definition 1 *Job $i \in N$ is ordinary, if*

$$\sum_{1 \le u \le \pi_1^{-1}(i)} b(\pi_1(u)) > \Omega. \tag{3.2.15}$$

Definition 2 *For every ordinary job $i \in N$ critical position $1 \le k_i \le n$ is defined as the smallest index v such that*

$$\sum_{1 \le u \le \pi_1^{-1}(i)} b(\pi_1(u)) - \sum_{1 \le u \le v} b(\pi_2(u)) \le \Omega. \tag{3.2.16}$$

Lemma 1 *If the pair of permutations (π_1, π_2) is feasible, then for each ordinary $i \in N$*

$$\pi_2^{-1}(i) > k_i. \tag{3.2.17}$$

Proof: Consider an arbitrary feasible schedule, in which the jobs are processed in the orders, specified by π_1 and π_2, and assume that $k_i > \pi_2^{-1}(i)$ for a job i. Consider all the jobs which are completed on the second-stage machine by the time $\tau = S_i^2 + p_i^2$. Since the considered schedule is feasible, $S_i^1 + p_i^1 \leq S_i^2$, hence $\{j:\ \pi_1^{-1}(j) \leq \pi_1^{-1}(i)\} \subseteq \{j:\ S_j^1 \leq \tau\}$. At the same time $\{j:\ S_j^2 + p_j^2 \leq \tau\} \subseteq \{j:\ \pi_2^{-1}(j) \leq \pi_2^{-1}(i)\}$. For the feasible schedule the capacity of the buffer Ω should not be exceeded in any moment of time, for example τ:

$$\sum_{u:\ S_u^1 \leq \tau} b(u) - \sum_{u:\ S_u^2 + p_u^2 \leq \tau} b(u) \leq \Omega. \tag{3.2.18}$$

Since $k_i > \pi_2^{-1}(i)$, by virtue of the definition 2 and (3.2.18),

$$\sum_{u:\ S_u^1 \leq \tau} b(u) - \sum_{u:\ S_u^2 + p_u^2 \leq \tau} b(u) \geq \sum_{1 \leq u \leq \pi_1^{-1}(i)} b(\pi_1(u)) - \sum_{1 \leq u \leq \pi_2^{-1}(i)} b(\pi_2(u)) > \Omega,$$

which contradicts (3.2.18). If $k_i = \pi_2^{-1}(i)$, then by the definition 2,

$$\sum_{1 \leq u < \pi_1^{-1}(i)} b(\pi_1(u)) - \sum_{1 \leq u \leq \pi_2^{-1}(i)} b(\pi_2(u)) \leq \Omega - b(i), \tag{3.2.19}$$

which implies that job i has to be processed on the second-stage machine and leave the buffer before it can start on the first-stage machine, which contradicts to the assumption that the considered schedule is feasible.□

Lemma 1 demonstrates that (3.2.17) is a necessary condition for feasibility of pair (π_1, π_2). To prove that (3.2.17) is also a sufficient feasibility condition, the results of the following lemma are required.

Lemma 2 *If for the pair (π_1, π_2) the inequality (3.2.17) holds for each ordinary $i \in N$, then for any q such that $\pi_2^{-1}(q) \leq k_i$, $\pi_1^{-1}(q) < \pi_1^{-1}(i)$.*

Proof: If q is not an ordinary job and $\pi_1^{-1}(q) \geq \pi_1^{-1}(i)$, then

$$\sum_{1 \leq u \leq \pi_1^{-1}(i)} b(\pi_1(u)) \leq \sum_{1 \leq u \leq \pi_1^{-1}(q)} b(\pi_1(u)) \leq \Omega,$$

which contradicts i being ordinary job. Assume that q is an ordinary job and $\pi_1^{-1}(q) \geq$

$\pi_1^{-1}(i)$. Then by virtue of (3.2.17), $k_q < \pi_2^{-1}(q) \leq k_i$. Thus, by the assumption and taking into account definition 2,

$$\sum_{1\leq u\leq \pi_1^{-1}(q)} b(\pi_1(u)) - \sum_{1\leq u\leq k_q} \geq b(\pi_2(u)) \sum_{1\leq u\leq \pi_1^{-1}(i)} b(\pi_1(q)) - \sum_{1\leq u\leq k_q} b(\pi_2(u)) > \Omega,$$

which contradicts to the fact that k_q is the critical position for job q. □

If a pair of permutations (π_1, π_2) is feasible the Algorithm WAIT described below constructs a schedule in which the order of execution of jobs on each machine coincides with permutations π_1 and π_2. Otherwise, the algorithm modifies the permutation π_2 in order to construct a feasible schedule. To determine, whether or not the pair π_1 and π_2 is feasible, the algorithm calculates a critical position k_i for all ordinary jobs $i \in N$ by going through permutation π_1 and determining whether or not (3.2.17) is satisfied for every job. Denote by t_1 and t_2 the current starting time on the first and the second-stage machine, correspondingly. Let pos_2 be the position of the last job in π_2, scheduled in the current iteration. Let $\xi^m(i)$, $m = \{1, 2\}$, be equal to 1 if the job i is already assigned to the machine m in the current schedule and 0 otherwise. Let $Calculate(\pi_1, \pi_2)$ be the procedure which calculates k_i for all ordinary jobs $i \in N$ and assigns $k_i = -1$ if job i is not ordinary. Further, $Calculate(\pi_1, \pi_2)$ sets $Pair(\pi_1, \pi_2) =$ **True**, if the pair is feasible, and $Pair(\pi_1, \pi_2) =$ **False**, otherwise. Let $Repair(i, pos_2)$ be the procedure which modifies the π_2 if the pair (π_1, π_2) is not feasible. For a permutation π assume that $|\pi|$ signifies the number of elements in π. The WAIT algorithm can be summarised as follows:

Algorithm WAIT

1: **Set** $\pi_1' = \pi_1$, $\pi_2' = \pi_2$, $t_1 = 0$, $t_2 = 0$ and $pos_2 = 0$.
2: $Calculate(\pi_1, \pi_2)$.
3: **while** $|\pi_1'| > 0$ **or** $|\pi_2'| > 0$ **do**
4: **if** $|\pi_1'| > 0$ **then**
5: **Set** $i = \pi_1'(1)$;
6: **if** $\xi^2(\pi_1'(k_i)) = 1$ **then**
7: **Set** $S_i^1 = \max\{t_1, S^2_{\pi_2'(k_i)} + p^2_{\pi_2'(k_i)}\}$; $t_1 = S_i^1 + p_i^1$, $\xi^1(i) = 1$;

Delete i from π_1'.
else
if NOT$(Pair(\pi_1, \pi_2))$ **then**
$Repair(i, pos_2)$.
end if
end if
end if
if $|\pi_2'| > 0$ **then**
Set $j = \pi_2'(1)$;
if $\xi^1(j) = 1$ **then**
Set $S_j^2 = \max\{t_2, S_{\pi_1'(j)}^2 + p_j^2\}$; $t_2 = S_j^2 + p_j^2$, $\xi^2(j) = 1$, $pos_2 = pos_2 + 1$;
delete j from π_2'.
end if
end if
end while

$Calculate(\pi_1, \pi_2)$ **procedure**

Set $Pair(\pi_1, \pi_2) = True$, $j = 1$.
while $j \leq n$ **and** $Pair(\pi_1, \pi_2)$ **do**
Set $i = \pi_1(j)$;
if $\sum_{1 \leq u \leq \pi_1^{-1}(i)} b(\pi_1(u)) \leq \Omega$, **then**
Set $k_i = -1$;
else
Set $k_i = \min\{v : \sum_{1 \leq u \leq \pi_1^{-1}(i)} b(\pi_1(u)) - \sum_{1 \leq u \leq v} b(\pi_2(u)) \leq \Omega\}$;
if $k_i \geq \pi_2(i)$ **then**
set $Pair(\pi_1, \pi_2) =$ **False.**
end if
end if
end while

Repair(i, pos_2) **procedure**

1: **Set** $r = 1$;
2: **if** $k_i < \pi_2^{-1}(i)$ **then**
3: **while** $r \leq k_i$ **and** $\pi_1^{-1}(\pi_2(r)) < \pi_1^{-1}(i)$ **do**
4: **Set** $r = r + 1$;
5: **end while**
6: **end if**(we have checked whether or not all jobs on positions in π_2 up to k_i are on positions less than $\pi_1^{-1}(i)$ in π_1)
7: **if** $k_i \geq \pi_2^{-1}(i)$ **or** $r \leq k_i$ **then**
8: **Set** $\gamma = pos_2$;
9: **while** $\sum_{1 \leq u \leq \pi_1^{-1}(i)} b(\pi_1(u)) - \sum_{1 \leq u \leq \gamma} b(\pi_2(u)) > \Omega$ **do**
10: **Set** $\mu = \min\{v > \gamma | \pi_2(v) = j \ \& \xi^1(j) = 1\}$;
11: **Set** $h = \pi_2(\mu)$, $\gamma = \gamma + 1$, $r = \mu$;
12: **for** $\gamma < r \leq \mu$ **do**
13: **Set** $\alpha = \pi_2(r-1)$, $\pi_2(r) = \alpha$
14: **end for**
15: **Set** $\pi_2(\gamma) = h$.
16: **end while**
17: **Set** $k_i = \gamma$;
18: **Set** $j = 1$;
19: **for** $j \leq |\pi_2'|$ **do**
20: **Set** $\pi_2'(j) = \pi_2(pos_2 + j)$ (here we are updating π_2').
21: **end for**
22: **end if**

Algorithm WAIT constructs a feasible schedule as lines 6, 9 of the algorithm and lines 7, 9 of the *Repair* procedure ensure that an ordinary job is scheduled on the first-stage machine only after the job on the corresponding critical position is completed on the second-stage machine, hence by definition of a critical position the buffer capacity is never violated; line 16 of the algorithm ensures that a job is scheduled on the second-stage machine only after its completion on the first-stage machine. Observe

that in *Repair* procedure the job on the position μ in line 10 of the procedure always exits, as otherwise all the jobs which have been already scheduled on the first-stage machine, would also have been scheduled on the second-stage machine before pos_2, which contradicts the line 9 of the *Repair* procedure. Please also note that if $\pi_1 = \pi_2$, than the pair (π_1, π_2) is feasible.

Description of Lagrangian Heuristics

Each of the proposed Lagrangian heuristics is an iterative procedure that utilises the subgradient method [22]. At each iteration, a Lagrangian relaxation for the current set of Lagrangian multipliers is solved and a feasible schedule is constructed using either NO-WAIT or WAIT algorithm. The objective value provided by the Lagrangian relaxation and the objective value provided by the feasible schedule are used to update the set of Lagrangian multipliers. It is convenient to use the following notation for the Lagrangian heuristics summary, which is provided below. Let $LowerBound(v, u)$ be a procedure which calculates the objective function of the Lagrangian relaxation according to (3.2.11) for the current set of Lagrangian multipliers (v, u). The Lagrangian relaxation provides starting times of jobs on each machine. These starting times determine the permutations of jobs π_1 and π_2 for the first and the second-stage machines, correspondingly. Essentially the WAIT and NO-WAIT Lagrangian heuristics are different only in the way a feasible schedule is constructed. Let $FEASIBLE(\pi_1, \pi_2)$ signify the value of the objective function provided by the feasible schedule constructed by either WAIT or NO-WAIT algorithm. Let $InitialUB$ be the value of the objective function provided by the feasible schedule constructed by either WAIT or NO-WAIT algorithm in non-increasing order of $W_i^0 = \frac{w_i}{(p_i^1 + p_i^2)}$, $i \in N$. Denote by NI_{max} the maximum number of iterations. Let X be the optimal solution vector, obtained during Lagrangian relaxation stage for the current set of (v, u), with coordinates $X_h = x_{it}^m$ for $0 \leq h < 2nT$, where $1 \leq i \leq n$, $0 \leq t < T$, $m \in \{1, 2\}$. Denote by A the matrix of the coefficients of the left hand sides of the dualised constraints (3.2.3) and (3.2.5), and by B the vector of the corresponding right hand sides. Let λ be a positive coefficient $0 < \lambda \leq 2$.

Lagrangian Heuristic

1: **Set** $v_{tm} = 0$ and $u_t = 0$ for $0 \leq t < T$, $m \in \{1, 2\}$; **set** $k = 0$.
2: **Set** $BestLB = 0$; $BestUB = InitialUB$.
3: **while** $k < NI_{max}$ **do**
4: $LB = LowerBound(v, u)$.
5: **if** $LB > BestLB$ **then**
6: $BestLB = LB$.
7: **end if**
8: $UB = FEASIBLE(\pi_1, \pi_2)$.
9: **if** $UB < BestUB$ **then**
10: $BestUB = UB$.
11: **end if**
12: **Set** $\tau = \lambda \frac{BestUB - LB}{\|AX - B\|^2}$.
13: $(v, u) = (v, u) + \tau(AX - B)$
14: **end while**

The above Lagrangian heuristic will be referred to as the NO-WAIT heuristic, if on the step 8 the NO-WAIT algorithm is used to obtain the current upper bound; and as to the WAIT heuristic, if on the step 8 the WAIT algorithm is used to obtain the current upper bound.

3.2.4 Computational experiments

The computational experiments aimed to compare the NO-WAIT and WAIT heuristics. The test instances were generated randomly with processing times chosen from the interval $[1, 10]$, and jobs' weights chosen from the interval $(0, 2]$. Each set consisted of 15 instances. An instance is described in the form $n - \Omega_k$, where n is the number of jobs, and Ω_k is the size of the buffer. The experiments were conducted for instances with 5, 10, 25 and 50 jobs and for buffer sizes $\Omega_1 = b_{max}$, $\Omega_{1.5} = 1.5b_{max}$ and $\Omega_2 = 2b_{max}$, where b_{max} is the maximum buffer requirement among all jobs of an instance. Recall that Lagrangian relaxation provides two permutations - π_1 and

π_2, defined by the starting times of jobs on the first and the second-stage machine, correspondingly. Denote by $NW1$ and $NW2$ the NO-WAIT heuristic, with the same orders of jobs on both machines defined by either π_1 or π_2 only, and by $NW3$ - the NO-WAIT heuristic, with the orders of jobs on first and second-stage machines defined by π_1 and π_2, correspondingly. Similar, denote by $W1$ and $W2$ the WAIT heuristic, with the same orders of jobs on both machines defined by either π_1 or π_2 only, and by $W3$ - the WAIT heuristic, with the orders of jobs on first and second-stage machines defined by π_1 or π_2, correspondingly. In each heuristic the subgradient algorithm was run for 1000 iterations, the time limit was 25 minutes for the small instances with 5 and 10 jobs, and 30 minutes for 25 and 50 jobs instances.

For each job i the parameter

$$W_i^0 = \frac{w_i}{(p_i^1 + p_i^2)}$$

was calculated, and the list of jobs in non-increasing order of W_i^0 was constructed. To obtain an initial value of the upper bound NO-WAIT or WAIT algorithm was employed with this order for both machines. Hence the initial upper bounds for $NW1$, $NW2$ and $NW3$ had the same value INW; similarly, the initial upper bounds for $W1$, $W2$ and $W3$ had the same value IW. "Swapping" the initial upper bounds by running WAIT algorithm with the initial upper bound INW and running NO-WAIT algorithm with the initial upper bound IW did not lead to significant changes in the resulting solutions, with the change of the values within 5%.

The Tables 3.1 - 3.12 compare the quality of the objective function values provided by WAIT and NO-WAIT heuristics and CPLEX (results by CPLEX are for 5 and 10 jobs instances only) and the tables are constructed as follows. The first column represents an instance number. For each instance the second column represents the difference D, in %, between the initial upper bounds provided by NO-WAIT and WAIT algorithms, and calculated as

$$D = \left(1 - \frac{IW}{INW}\right) \times 100\%.$$

The columns $3-5$ show the improvement, in %, of the upper bound I, calculated as

$$I = \left(1 - \frac{UB}{INW}\right) \times 100\%,$$

where UB is a value of the objective function provided by a feasible schedule, constructed by $NW1$, $NW2$ and $NW3$ correspondingly. The columns $6-8$ show the improvement, in %, of the upper bound I, calculated as

$$I = \left(1 - \frac{UB}{IW}\right) \times 100\%,$$

where UB is a value of the objective function provided by a feasible schedule, constructed by $W1$, $W2$ and $W3$ correspondingly. Columns $9-15$ show the deviation of the objective value UB, provided by a heuristic ($NW1$, $NW1$, $NW2$, $NW3$, $W1$, $W2$, $W3$ and CPLEX - for 5 and 10 jobs sets) from the best value among the heuristics, calculated as

$$\left(\frac{UB}{BestValue} - 1\right) \times 100\%.$$

Observe that more than one heuristic can provide the best value.

For small instances of 5 and 10 jobs all heuristics provided optimal/near-optimal solutions for most instances, compared with the solutions obtained by CPLEX software.

For most instances across all numbers of jobs and buffer sizes the heuristic $W1$ provided the best values, which indicates that the order of starting times on the first-stage machine, obtained during the Lagrangian relaxation stage, is more significant, than the order on the second-stage machine. Moreover, WAIT heuristics $W1$ and $W2$ provided smaller upper bounds, than NO-WAIT and $W3$ heuristics for most of the instances. Observe that WAIT algorithm also provided the smallest initial upper bound for most instances.

The graphs on Figures 3-1 - 3-2 illustrate how the upper and lower bounds change with each iteration for instances $50-\Omega_2$ and $25-\Omega_2$ for each heuristic. At each iteration the upper bound UB is the smallest value of the objective function so far,

Table 3.1: 5 jobs instances, buffer size $\Omega_{1.0}$

	Initial UB, 1-IW/INW,%	Improvement, 1-UB/INW, %			Improvement, 1-UB/IW, %			Best UB, UB/Best UB -1, %						
Instance	D ,%	NW1	NW2	NW3	W1	W2	W3	NW1	NW2	NW3	W1	W2	W3	CPLEX
1	0	0	0	0	0	0	0.0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
2	-3	0	0	0	**3**	**3**	**3**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
3	14	**15**	14	**15**	**2**	0	0.4	**0**	2	**0**	**0**	2	2	**0**
4	-3	**1**	**1**	**1**	**3**	**3**	**3**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
5	5	0	0	0	0	0	0.3	5	5	5	**0**	**0**	**0**	**0**
6	0	0	0	0	0	0	0.0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
7	0	0	0	0	0	0	0.0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
8	-2	0	0	0	**2**	**2**	**2**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
9	8	**6**	1	**6**	**1**	**1**	0.1	4	9	4	**0**	**0**	1	**0**
10	6	0	0	0	0	0	0	7	7	7	**0**	**0**	**0**	**0**
11	-4	0	0	0	**4**	**4**	**4**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
12	6	**4**	**4**	**4**	**4**	**4**	**4**	7	7	7	**0**	**0**	**0**	**0**
13	0	0	0	0	0	0	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
14	23	**10**	0	**10**	0	0	0	18	30	18	**0**	**0**	**0**	**0**
15	0	0	0	0	0	0	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Table 3.2: 5 jobs instances, buffer size $\Omega_{1.5}$

	Initial UB, 1-IW/INW,%	Improvement, 1-UB/INW, %			Improvement, 1-UB/IW, %			Best UB, UB/Best UB -1, %						
Instance	D ,%	NW1	NW2	NW3	W1	W2	W3	NW1	NW2	NW3	W1	W2	W3	CPLEX
1	0	**0.1**	0	**0.1**	**0.1**	0	0	**0**	0.1	**0**	**0**	0.1	0.1	**0**
2	-9	0	0	0	**8**	**8**	**8**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
3	18	**28**	23	**28**	**13**	**13**	**13**	2	8	2	**0**	**0**	**0**	**0**
4	0	0	0	0	0	0	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
5	10	0	0	0	**11**	**11**	**11**	26	26	26	**0**	**0**	**0**	**0**
6	0	0	0	0	0	0	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
7	13	**19**	**19**	**19**	**7**	**7**	**7**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
8	0	0	0	0	0	0	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
9	3	0	0	0	0	0	0	3	3	3	0.4	**0**	0.4	**0**
10	0	0	0	0	0	0	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
11	0	0	0	0	0	0	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
12	-10	**1**	**1**	**1**	**10**	**10**	**10**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
13	-7	0	0	0	**6**	**6**	**6**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
14	-1	0	0	0	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
15	-2	0	0	0	0	**2**	0	**0**	**0**	**0**	2	**0**	2	**0**

Table 3.3: 5 jobs instances, buffer size $\Omega_{2.0}$

	Initial UB, 1-IW/INW,%	Improvement, 1-UB/INW, %			Improvement, 1-UB/IW, %			Best UB, UB/Best UB -1, %						
Instance	D ,%	NW1	NW2	NW3	W1	W2	W3	NW1	NW2	NW3	W1	W2	W3	CPLEX
1	0	0	0	0	0	0	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
2	0	0	0	0	**0.5**	**0.5**	**0.5**	1	1	1	**0**	**0**	**0**	**0**
3	0	**8**	**8**	**8**	**8**	**8**	7	**0**	**0**	**0**	**0**	**0**	1	**0**
4	0	**7**	**7**	**7**	**7**	**7**	0	**0**	**0**	**0**	**0**	**0**	8	**0**
5	-1	0	0	0	**5**	**5**	**5**	4	4	4	**0**	**0**	**0**	**0**
6	0	0	0	0	0	0	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
7	0	6	6	6	6	6	0	**0**	**0**	**0**	**0**	**0**	7	**0**
8	0	0	0	0	0	0	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
9	5	0	0	0	0	0	0	5	5	5	**0**	**0**	**0**	**0**
10	0	0	0	0	**0.05**	**0.05**	**0.05**	0.05	0.05	0.05	**0**	**0**	**0**	**0**
11	0	**2**	**2**	**2**	**2**	**2**	**2**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
12	0	**5**	**5**	**5**	**8**	**8**	**8**	3	3	3	**0**	**0**	**0**	**0**
13	0	0	0	0	**1**	**1**	0	1	1	1	**0**	**0**	0.9	**0**
14	3	0	**1**	0	**0.3**	0	0	3	2	3	**0**	0.3	0.3	**0**
15	6	0	0	0	0	0	0	6	6	6	**0**	**0**	**0**	**0**

Table 3.4: 10 jobs instances, buffer size $\Omega_{1.0}$

	Initial UB, 1-IW/INW,%	Improvement, 1-UB/INW, %			Improvement, 1-UB/IW, %			Best UB, UB/Best UB -1, %						
Instance	D ,%	NW1	NW2	NW3	W1	W2	W3	NW1	NW2	NW3	W1	W2	W3	CPLEX
1	-3	**2**	1	**2**	**4**	**4**	**4**	**0**	0.2	**0**	0.2	0.2	0.2	**0**
2	0	0	0	0	0	0	0	**0**	**0**	**0**	0.4	0.4	0.4	99
3	-5	0	0	0	**7**	**7**	**7**	2	2	2	0.02	0.02	0.02	**0**
4	4	5	**7**	5	6	**10**	3	9	7	9	4	**0**	7	263
5	-4	5	**6**	5	**10**	**10**	**10**	1	**0**	1	0.1	0.1	0.1	**0**
6	5	**11**	**11**	**11**	8	7	**8**	2	2	2	0.4	1	**0**	**0**
7	-7	**4**	**4**	**4**	**10**	**10**	**10**	**0**	**0**	**0**	**0**	**0**	**0**	1
8	-4	**6**	**6**	**6**	9	**10**	5	0.4	0.4	0.4	1	**0**	5	**0**
9	-8	**0**	**0**	**0**	8	**9**	8	1	1	1	0.5	**0**	1	0.5
10	-3	**6**	1	5	**13**	10	9	6	12	7	1	4	6	**0**
11	-2	4	**4**	4	**7**	7	7	2	1	2	**0**	0.1	0.2	112
12	2	7	7	7	**10.0**	**10.0**	9.7	4	4	4	**0**	**0**	0.3	2
13	8	5	5	**5**	**2**	**2**	1	6	6	6	0.03	0.03	1	**0**
14	9	**16**	14	**16**	**7**	**7**	6	**0**	2	**0**	1	1	2	**0**
15	-1	4	**6**	4	7	**8**	7	3	1	3	1	**0**	1	1

Table 3.5: 10 jobs instances, buffer size $\Omega_{1.5}$

	Initial UB, 1-IW/INW,%	Improvement, 1-UB/INW, %			Improvement, 1-UB/IW, %			Best UB, UB/Best UB -1, %						
Instance	D ,%	NW1	NW2	NW3	W1	W2	W3	NW1	NW2	NW3	W1	W2	W3	CPLEX
1	-5	**13**	**13**	**13**	**17**	16	15	0.2	0.2	0.2	0.2	2	2	**0**
2	-14	**4**	2	**4**	**14**	11	9	**0**	2	**0**	2	5	7	2595
3	8	10	9	**12**	**7**	5	0	4	6	3	0.4	2	8	**0**
4	26	**32**	29	29	8	**10**	2	2	7	7	2	1	9	**0**
5	4	**3**	**3**	**3**	**3**	**3**	2	5	5	5	1	1	1	**0**
6	0.1	**2**	0.1	0.1	**6**	5	5	6	7	7	1	2	2	**0**
7	-10	0.6	**1.0**	0.6	**13**	10	7	4	3	4	**0**	3	6	2435
8	-8	**4**	2	2	**11.7**	**12.1**	7.2	1	4	3	1	0.4	6	**0**
9	13	12	**12**	**12**	**4**	3	2	7	6	6	1	3	4	**0**
10	11	**20**	18	18	**8**	**8**	2	0.1	3	3	1	1	9	**0**
11	1	**6**	**6**	**6**	**9**	8	**9**	5	5	5	1	2	1	**0**
12	13	8	10.7	**11.0**	**2.3**	1.9	0.9	8	5	5	**0**	0.4	1	**0**
13	3	**0.2**	**0.2**	**0.2**	**3**	**3**	2	6	6	6	0.05	0.05	0.4	**0**
14	15	**26**	14	24	**13**	9	8	**0**	16	3	**0**	5	6	**0**
15	9	7.0	**14.2**	14.0	6.80	**6.82**	2.6	9	1	1	0.02	**0**	5	1

Table 3.6: 10 jobs instances, buffer size $\Omega_{2.0}$

	Initial UB, 1-IW/INW,%	Improvement, 1-UB/INW, %			Improvement, 1-UB/IW, %			Best UB, UB/Best UB -1, %						
Instance	D ,%	NW1	NW2	NW3	W1	W2	W3	NW1	NW2	NW3	W1	W2	W3	CPLEX
1	3	**18**	**18**	**18**	**15**	**15**	12	0.2	0.2	0.2	0.2	0.2	5	**0**
2	14	**12**	6	**12**	**3**	3	1	6	13	6	**0**	0.001	1	**0**
3	-2	**2**	**2**	**2**	**7.1**	6.8	6.6	3	3	3	**0**	0.3	0.5	**0**
4	-3	0	**1**	0	3	**4**	3	1	**0**	1	1	**0**	1	**0**
5	0	**3**	**3**	**3**	**4**	**4**	**4**	1	1	1	**0**	**0**	**0**	**0**
6	0.3	**0.3**	**0.3**	0.2	**2.2**	**2.2**	2.1	2	2	2	**0**	**0**	0.1	**0**
7	-2	**2**	**2**	**2**	**4**	**4**	0.7	**0**	**0**	**0**	**0**	**0**	3	**0**
8	-1	**7**	**7**	6	7	**9**	5	0.3	0.3	2	1	**0**	4	**0**
9	0	1.5	**1.6**	1.5	**2**	**2**	**2**	0.2	0.1	0.2	**0**	**0**	**0**	**0**
10	5	**11**	8	**11**	**8.1**	7.7	6.2	4	7	4	2	2	4	**0**
11	1	**12**	**12**	**12**	12.200	**12.201**	12.200	2	2	2	0.002	**0**	0.002	**0**
12	8	9.8	9.6	**10.0**	**2**	**2**	**2**	0.4	1	0.1	**0**	**0**	**0**	**0**
13	3	**5**	4	4	**2**	1	1	**0**	0.5	0.5	0.1	0.5	1	**0**
14	6	**10**	**10**	**10**	**5**	**5**	**5**	1	1	1	**0**	**0**	**0**	**0**
15	3	**11**	6	9	**11**	**11**	10	3	9	6	1	1	2	**0**

Table 3.7: 25 jobs instances, buffer size $\Omega_{1.0}$

	Initial UB, 1-IW/INW,%	Improvement, 1-UB/INW, %			Improvement, 1-UB/IW, %			Best UB, UB/Best UB -1, %					
Instance	D ,%	NW1	NW2	NW3	W1	W2	W3	NW1	NW2	NW3	W1	W2	W3
1	18	14	14	**19**	**11**	9	0	18	18	10	**0**	2	12
2	12	**14**	13	13	**10**	9	0	8	9	10	**0**	1	11
3	0.5	**9**	8	8	11	**13**	7	5	6	6	2	**0**	6
4	10	**16.1**	16.0	15.7	**11**	8	1	4	5	5	**0**	3	11
5	6	**15.0**	13	14.8	13.00	**13.03**	5	4	7	5	0.03	**0**	9
6	2	12	11	**13**	13.126	**13.129**	10.7	3	5	2	**0**	**0**	3
7	8	13	13	**14**	**7**	5	1	1	1	1	**0**	2	6
8	6	8	3	**8**	**15**	14	10	16	21	16	**0**	1	5
9	9	6	**10**	6	6.4	**6.8**	1.6	10	6	10	0.4	**0**	6
10	8	10	10	**14**	**11**	9	5	10	9	4	**0**	2	6
11	-13	**4**	2	4	11.6	**12.0**	9	**0**	1	0.0	3	3	6
12	-2	6	**8**	7	**14.3**	12.9	6	8	5	6	**0**	2	9
13	12	**16.49**	15	16.48	14.3	**14.5**	11	12	13	12	0.2	**0**	4
14	2	4.2	4.5	**4.7**	4	**6**	1	4	4	3	2	**0**	6
15	2	**6.4**	4	5.6	**5**	4	1	0.4	3	1	**0**	1	4

Table 3.8: 25 jobs instances, buffer size $\Omega_{1.5}$

	Initial UB, 1-IW/INW,%	Improvement, 1-UB/INW, %			Improvement, 1-UB/IW, %			Best UB, UB/Best UB -1, %					
Instance	D ,%	NW1	NW2	NW3	W1	W2	W3	NW1	NW2	NW3	W1	W2	W3
1	22	**18**	17	15	**9**	7	0	16	16	19	**0**	2	10
2	10	14	**16**	13	**9.4**	8.7	1	6	2	6	**0**	1	10
3	7	10	10	**11**	**9**	7	0	6	6	5	**0**	2	10
4	6	**16**	14	13	11	**13**	0	3	5	6	2	**0**	15
5	5	15	**17**	15	**13.2**	12.9	12	3	1	3	**0**	0.4	2
6	7	9	**12**	9	5	**5**	0	3	0.1	3	1	**0**	5
7	1	**16**	15	15	12	**14**	1	**0**	2	1	4	2	17
8	-2	**6**	5	4	**18**	17	11	13	14	15	**0**	2	9
9	12	**17**	15	12	**12**	10	0	7	9	14	**0**	2	14
10	13	**24**	22	22	**17**	16	8	5	7	8	**0**	1	11
11	6	13	12	**14**	**6**	5	0	1	3	**0**	2	4	9
12	10	**17.2**	16.9	15.9	6	**7**	0	**0**	0.3	2	2	1	9
13	20	19.1	19.5	**19.6**	9	**12**	0	15	15	15	4	**0**	14
14	16	**19**	7	18	11	**11**	0	9	24	9	1	**0**	13
15	2	12.6	**12.7**	12.6	13	**16**	4	7	7	7	3	**0**	15

Table 3.9: 25 jobs instances, buffer size $\Omega_{2.0}$

	Initial UB, 1-IW/INW,%	Improvement, 1-UB/INW, %			Improvement, 1-UB/IW, %			Best UB, UB/Best UB -1, %					
Instance	D ,%	NW1	NW2	NW3	W1	W2	W3	NW1	NW2	NW3	W1	W2	W3
1	19	**18**	12	15	5	**8**	0	9	17	13	3	**0**	8
2	2	6	**9**	7	**8**	7	2	5	2	4	**0**	1	7
3	4	**11.10**	10.8	11.08	7	**8**	2	0.2	0.5	0.2	1	**0**	6
4	3	**12**	3	10	11	**12**	5	3	13	5	0.3	**0**	7
5	10	19.4	19.5	**19.6**	12.7	**13.2**	5	3	3	3	1	**0**	10
6	1	7.8	**8.1**	7.8	7.6	**8.1**	2	1	1	1	0.4	**0**	6
7	7	13.02	**13.08**	13.06	**7.8**	7.6	0	1	1	1	**0**	0.3	8
8	8	14	13	**16**	**15**	14	13	9	11	7	**0**	1	2
9	9	11.38	11.26	**11.43**	**7**	6	0	5	6	5	**0**	1	8
10	9	**19.3**	18.6	18	13	**15**	3	4	5	5	2	**0**	13
11	-0.2	7	8	7	6	**7**	0	0.5	**0**	1	1	0.5	8
12	6	**13**	12	12	**7.0**	6.9	0	0.1	1	1	**0**	0.1	8
13	12	**17**	15	11	11	**12**	0	8	10	15	1	**0**	14
14	19	**24**	21	24	**8**	6	2	2	6	2	**0**	2	7
15	5	**17.3**	16.5	17.0	**13**	12	8	1	2	1	**0**	2	6

Table 3.10: 50 jobs instances, buffer size $\Omega_{1.0}$

	Initial UB, 1-IW/INW,%	Improvement, 1-UB/INW, %			Improvement, 1-UB/IW, %			Best UB, UB/Best UB -1, %					
Instance	D ,%	NW1	NW2	NW3	W1	W2	W3	NW1	NW2	NW3	W1	W2	W3
1	9	**11.0**	10.5	8	**7**	6	0	4	5	8	**0**	1	7
2	7	11.6	**12.0**	12	**15**	15	6	12	11	12	**0**	1	11
3	16	10	5	**10**	5	**7**	0	15	21	15	2	**0**	7
4	21	**16**	19	16	**8**	5	0	16	12	16	**0**	4	9
5	-7	3	3.8	**4.2**	**15.4**	15.1	7	8	6	6	**0**	0.4	10
6	-4	**3**	1	2	**9**	8	0	2	4	4	**0**	1	10
7	6	4	4	**11**	**11**	10	0	14	14	7	**0**	1	12
8	13	**15**	11	11	**10**	9	0	9	14	13	**0**	1	11
9	-5	9	6	**10**	**15**	14	6	3	7	2	**0**	1	11
10	10	**9**	6	7	**5**	3	0	7	10	9	**0**	2	5
11	4	8	**11**	10	10	**10**	0	7	4	5	0.3	**0**	11
12	12	11	8	**13**	**12**	9	0	15	18	12	**0**	3	13
13	14	7	6	**7**	**6**	6	0	16	17	16	**0**	0.5	7
14	6	**10**	7	9	**11**	10	0	7	10	8	**0**	1	12
15	13	17.6	14	**17.8**	**12**	8	0	8	13	8	**0**	5	13

Table 3.11: 50 jobs instances, buffer size $\Omega_{1.5}$

	Initial UB, 1-IW/INW,%	Improvement, 1-UB/INW, %			Improvement, 1-UB/IW, %			Best UB, UB/Best UB -1, %					
Instance	D ,%	NW1	NW2	NW3	W1	W2	W3	NW1	NW2	NW3	W1	W2	W3
1	-4	0	3	**4**	**8**	7	0	5	2	**0**	1	2	9
2	18	**16**	13	16	**12**	11	0	17	22	17	**0**	1	14
3	12	**21**	18	18	**10.1**	9.8	0	**0**	4	4	1	1	12
4	23	15	15.55	**15.62**	**6**	4	0	17	16	16	**0**	2	6
5	-3	**7**	6	5	**10**	7	0	1	2	2	**0**	3	11
6	8	**14**	12	13	**11**	8	0	5	7	7	**0**	3	12
7	4	7	4	**7**	**14**	13	0	12	17	12	**0**	1	16
8	13	**22**	19	19	**11**	10	0	**0**	5	4	0.1	1	12
9	15	20	21	**21**	**5**	5	0	2	0.4	**0**	2	2	8
10	9	**14**	11	10	**8**	4	0	3	7	8	**0**	3	8
11	7	20	17	**20**	**12**	10	0	0	5	**0**	3	5	17
12	14	**20**	19	19	**13**	12	0	6	8	8	**0**	2	15
13	12	17	**19**	18	13	**14**	0	9	6	8	1	**0**	16
14	17	20	20	**21**	**13**	11	0	11	10	9	**0**	2	15
15	20	17	**19**	17	**5**	3	0	9	8	9	**0**	3	5

Table 3.12: 50 jobs instances, buffer size $\Omega_{2.0}$

	Initial UB, 1-IW/INW,%	Improvement, 1-UB/INW, %			Improvement, 1-UB/IW, %			Best UB, UB/Best UB -1, %					
Instance	D ,%	NW1	NW2	NW3	W1	W2	W3	NW1	NW2	NW3	W1	W2	W3
1	-1	**5.4**	4.8	4.5	6	**7**	0	1	1.9	2	2	**0**	8
2	24	**23**	17	19	**9**	7	0	11	20	16	**0**	2	10
3	14	**18**	13	16	**9**	8	0	5	11	6	**0**	1	9
4	19	**19.1**	19.0	17	**5**	5	0	4	4	7	**0**	0.1	5
5	10	17.9	17.6	**18.2**	**9**	9	0	1	1	1	**0**	1	10
6	7	**13**	11	12	**9**	8	0	3	5	4	**0**	1	10
7	17	**17**	14	16	10	**10**	0	12	15	13	0.02	**0**	12
8	6	**11**	7	8	**5**	5	0	1	5	4	**0**	1	6
9	11	**15.0**	11	14.6	**7**	6	0	3	8	3	**0**	1	8
10	19	**20**	13	16	2	**3**	0	1	10	6	0.1	**0**	3
11	4	**13**	12	13	9	**9**	0	**0**	2	1	2	1	11
12	20	**23**	20	21	**9**	8	0	7	10	9	**0**	1	10
13	16	**18**	10	15	**10**	8	0	9	20	13	**0**	2	11
14	16	**21**	20	19	**7**	7	0	1	3	4	**0**	0.5	8
15	18	**20.24**	20.18	18	**6**	5	0	3	3	6	**0**	0.2	6

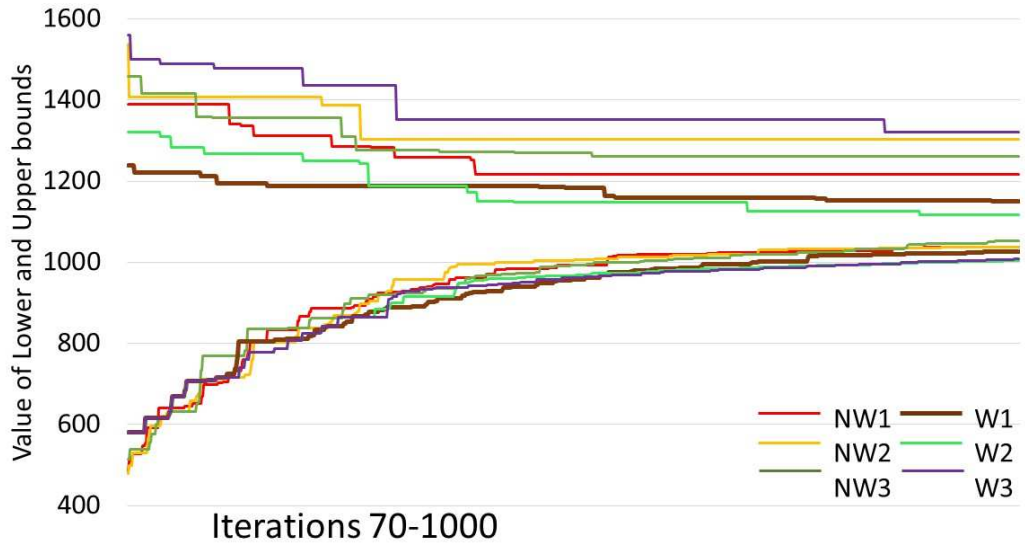
which is provided by a feasible schedule, constructed by corresponding WAIT or NO-WAIT algorithm; and the lower bound LB is the largest value of the objective function of the Lagrangian relaxation (3.2.11) so far. Clearly, WAIT heuristics $W1$ and $W2$ have the smallest relative errors RE, in %, which is calculated as

$$RE = \frac{UB - LB}{UB} \times 100\%.$$

The relative error was calculated for each instance within each set - the results are represented by box-plot charts on Figures 3-3 - 3-4. Relative errors for larger buffer size Ω_2 is smaller for all heuristics; relative errors provided by $W1$ and $W2$ are considerably smaller than the relative errors provided by $W3$ and NO-WAIT heuristics and the values have less variability across each set of instances.

In terms of average CPU time required to process an instance, both WAIT and NO-WAIT heuristics surpass direct integer programming approach - as illustrated on Figure 3-5, where the average CPU time in seconds is shown for instances of 5, 10, 15, 25 and 50 jobs and the buffer size Ω_2. If for the instances with 5 and 10 jobs the times are comparable, for the instances of 15 jobs the average time to obtain an optimal solution is already considerably greater. CPLEX failed to obtain a solution for a 15 jobs instance with buffer Ω_1 within 3 hours limit; for a 25 jobs instance with

Figure 3-1: Upper and Lower bounds change with iterations: 25 jobs



buffer Ω_2 CPLEX failed to obtain a solution within 10 hours limit.

In summary, the results demonstrate that WAIT heuristics, which follow the order of jobs provided during the Lagrangian relaxation stage, produce feasible schedules with a smaller value of the objective function, than NO-WAIT heuristics. It appears that the order of jobs on the first-stage is more significant, as WAIT heuristic $W1$, in which the jobs are scheduled in each machine exactly in the order, obtained by the Lagrangian relaxation and defined by starting times of jobs on the first-stage machine, provided the smallest value of the objective function for most instances, and had smaller relative errors. All Lagrangian heuristics provide feasible solutions in a much shorter time than CPLEX.

3.2.5 Conclusion

This section is concerned with Lagrangian relaxation and decomposition-based approach applied to the two-stage flow shop problem with a job-dependent buffer and the objective of minimisation of the total weighted completion time. The buffer re-

Figure 3-2: Upper and Lower bounds change with iterations: 50 jobs

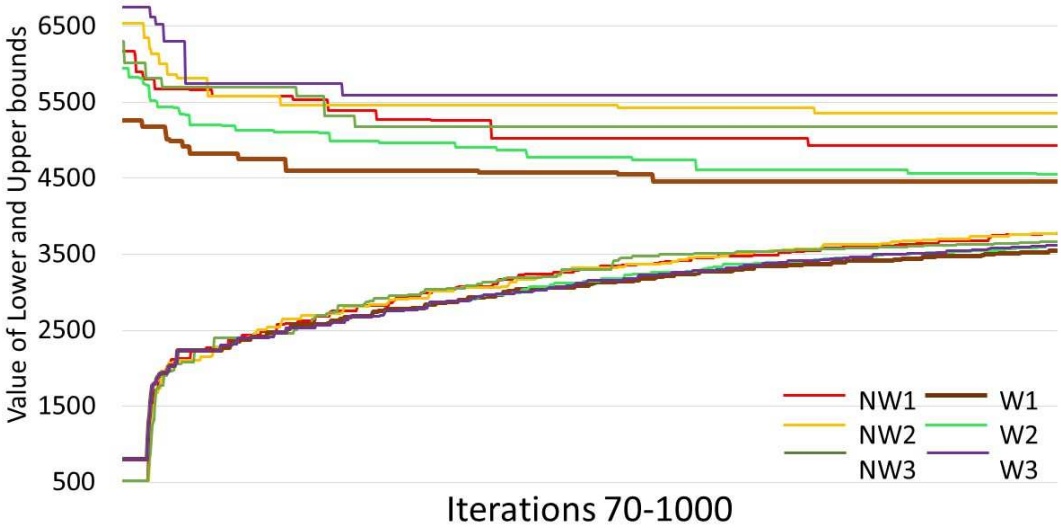


Figure 3-3: Relative error for 25 job instances, in %

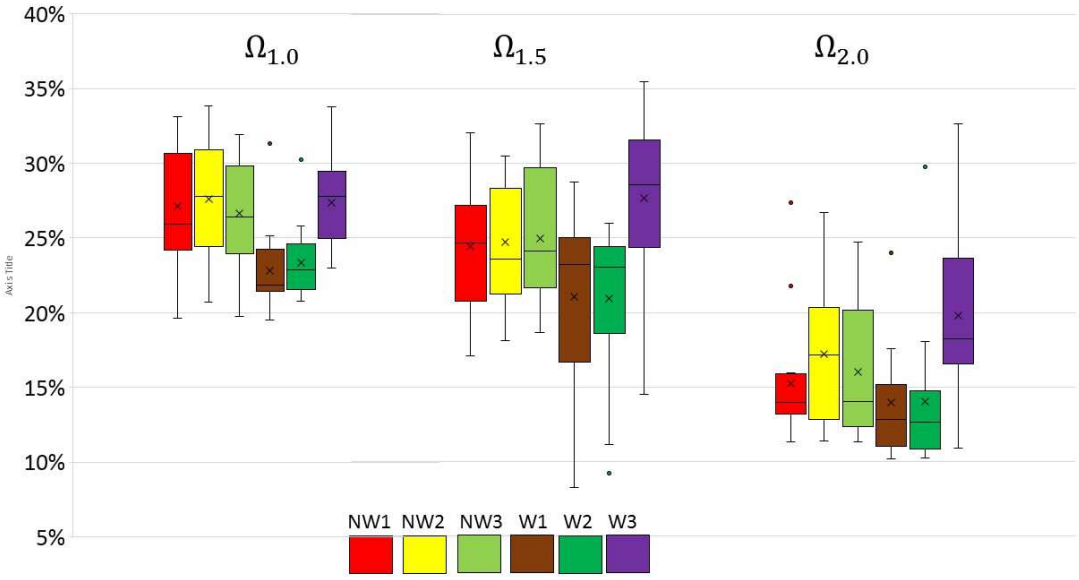


Figure 3-4: Relative error for for 50 job instances, in %

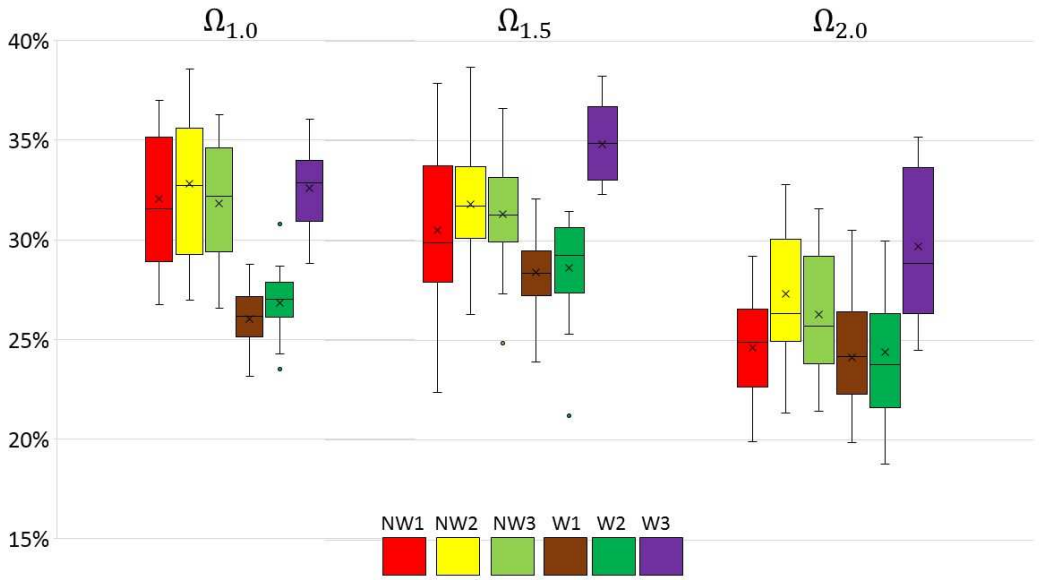
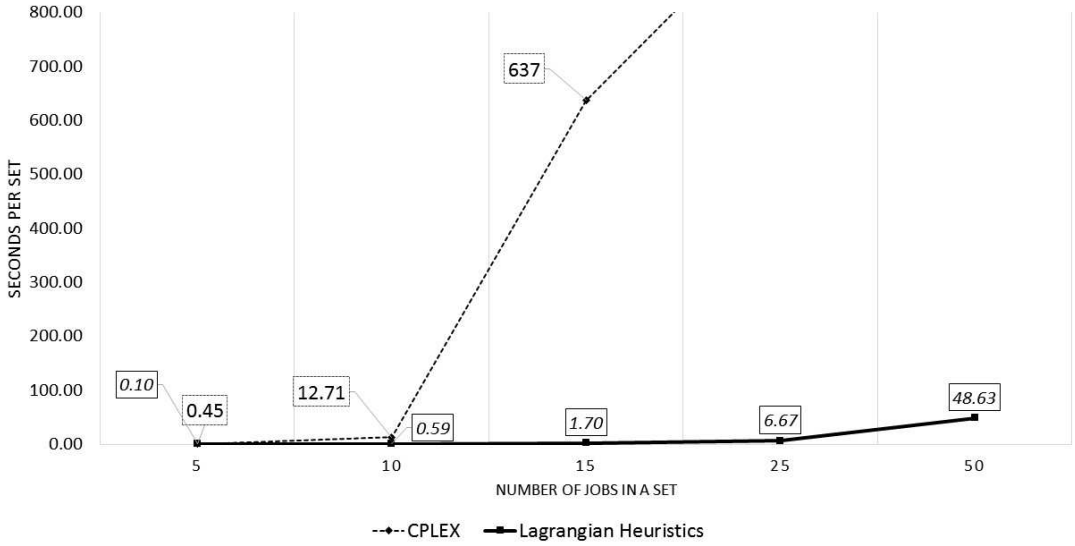


Figure 3-5: Average CPU time for instance with different number of jobs



quirement in this scheduling model varies from job to job and a job occupies the buffer continuously from the start of its first operation till the completion of its second operation. The Lagrangian relaxation and decomposition-based approach utilises a fast recursive algorithm for each subproblem, obtained by this decomposition. The resulting WAIT and NO-WAIT Lagrangian heuristics are compared by the means of computational experiments. The computational experiments demonstrate that the heuristics' results surpass the straightforward application of CPLEX optimisation software.

3.3 Two-stage flow shop with storage and objective to minimise maximum completion time

3.3.1 Problem Description

In what follows, the Lagrangian relaxation and decomposition are applied to the following problem. A set of jobs $N = \{1, ..., n\}$ is to be processed by two machines - the first-stage machine and the second-stage machine. Each job i is processed on the first-stage machine during p_i^1 (the first operation of the job) and on the second-stage machine during p_i^2 (the second operation of the job) time units. All processing times are integer. The second operation of a job can commence on the second-stage machine only after the completion of its first operation on the first-stage machine. Once an operation has started, it cannot be interrupted, i.e. no preemptions are allowed. Each machine can process at most one job at a time, and each job can be processed by at most one machine at a time. The processing of jobs commences at time $t = 0$. To be processed, each job i requires $b(i)$ units of the buffer space. This buffer space is occupied by a job continuously from the start of its first operation till the completion of its second operation. At any point in time t, the total buffer requirement of all jobs that started their processing before or at t and have a completion time of their second operation greater than t, can not exceed Ω - the buffer capacity. Again, similar to [59], [62], [73], [74] and [75] it is assumed that the buffer requirement of each job is determined by the duration of the first operation: $b(i) = p_i^1$ for all $i \in N$. A schedule σ specifies for each $j \in N$ the points in time $S_j^1(\sigma)$ and $S_j^2(\sigma)$, when job j starts processing, and $C_j^1(\sigma)$ and $C_j^2(\sigma)$, when job j completes processing on the first and the second-stage machine, correspondingly. Thus $S_j^1(\sigma) + p_j^1 = C_j^1(\sigma)$ and $S_j^2(\sigma) + p_j^2 = C_j^2(\sigma)$. The goal is to minimise the makespan $C_{max}(\sigma) = \max_{j \in N} C_j^2(\sigma)$. The considered problem is NP-hard in a strong sense [73].

3.3.2 Lagrangian relaxation-based heuristic

Integer Programming formulation

Denote by T the planning horizon, i.e. T is a non-negative number such that for an optimal schedule σ the value of makespan $C_{max}(\sigma) \leq T$.

One obvious choice for the planning horizon T is $\sum_{i \in N}(p_i^1 + p_i^2)$. However, a smaller planning horizon T may improve the convergence of an algorithm. To obtain a tighter T, WAIT algorithm (from section 3.2) is run with the permutation defined by non-increasing order of $p_i^1 + p_i^2$, and then T is set to the resulting value of the makespan. Define x_{it}^m, $i \in N$, $0 \leq t < T$, $m \in \{1, 2\}$, as

$$x_{it}^m = \begin{cases} 1, & \text{if } S_i^m = t; \\ 0, & \text{otherwise.} \end{cases}$$

Denote by $C_{max} = \max_{i \in N} \sum_{t=1}^{T-1} t x_{it}^2 + p_i^2$. The considered scheduling problem can be formulated as:

$$\min \ C_{max} \tag{3.3.1}$$

subject to

$$\sum_{t=0}^{T-1} x_{it}^m = 1, \quad \text{for } 1 \leq i \leq n \text{ and } m \in \{1, 2\} \tag{3.3.2}$$

$$\sum_{i=1}^{n} \sum_{\tau=\max\{0, t-p_i^m+1\}}^{t} x_{i\tau}^m \leq 1, \quad \text{for } 0 \leq t < T, \ \ m \in \{1, 2\} \tag{3.3.3}$$

$$\sum_{t=1}^{T-1} t x_{it}^2 - \sum_{t=1}^{T-1} t x_{it}^1 \geq p_i^1, \quad \text{for } 1 \leq i \leq n \tag{3.3.4}$$

$$\sum_{i=1}^{n} b(i) \left(\sum_{\tau=0}^{t} x_{i\tau}^1 - \sum_{\tau=0}^{t-p_i^2} x_{i\tau}^2 \right) \leq \Omega, \quad \text{for } 0 \leq t < T \tag{3.3.5}$$

$$\sum_{t=1}^{T-1} t x_{it}^2 + p_i^2 \leq C_{max}, \quad \text{for } 1 \leq i \leq n \tag{3.3.6}$$

$$x_{it}^m \in \{0, 1\}, \quad \text{for } 1 \leq i \leq n, \ 0 \leq t < T, \ m \in \{1, 2\}; \ C_{max} \geq 0 \tag{3.3.7}$$

The constraints (3.3.2) signify that each job can start only once on each machine, the constraints (3.3.3) imply that only one operation is processed on each machine

at a time, the constraints (3.3.4) ensure the correct order of operations for each job, the constraints (3.3.5) enforce that the overall buffer capacity is not exceeded at any time, (3.3.6) define the value of C_{max} and the constraints (3.3.7) is non-negativity constraint.

Lagrangian relaxation and decomposition

To obtain the Lagrangian relaxation the capacity constraints (3.3.3) and (3.3.5) are relaxed. To relax (3.3.6) the technique described in [102] is utilised: for multipliers $\lambda_i \geq 0$ with at least one $\lambda_j > 0$, the constraints (3.3.6) are aggregated:

$$\sum_{i=1}^{n} \lambda_i \left(\sum_{t=1}^{T-1} t x_{it}^2 + p_i^2 \right) \leq \sum_{i=1}^{n} \lambda_i C_{max}$$

or

$$\sum_{i=1}^{n} \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j} \left(\sum_{t=1}^{T-1} t x_{it}^2 + p_i^2 \right) \leq C_{max}. \tag{3.3.8}$$

Denote by $q_i = \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j}$, $i \in N$, and let $v_{tm} \geq 0$ and $u_t \geq 0$, $m \in \{1, 2\}$, $0 \leq t < T$, be Lagrangian multipliers. Then the following Lagrangian relaxation is obtained:

$$\begin{aligned}
&\min\ C_{max} + \sum_{i=1}^{n} q_i \left(\sum_{t=1}^{T-1} t x_{it}^2 + p_i^2 \right) - C_{max} \\
&+ \sum_{t=0}^{T-1} \left[\sum_{m=1}^{2} v_{tm} \left(\sum_{i=1}^{n} \sum_{\tau=\max\{0, t-p_i^m+1\}}^{t} x_{i\tau}^m - 1 \right) \right] \\
&+ \sum_{t=0}^{T-1} u_t \left[\sum_{i=1}^{n} b(i) \left(\sum_{\tau=0}^{t} x_{i\tau}^1 - \sum_{\tau=0}^{t-p_i^2} x_{i\tau}^2 \right) - \Omega \right]
\end{aligned}$$

subject to (3.3.2), (3.3.4) and (3.3.7). Let (v, u, q) be the sets of all Lagrangian multipliers, and $L(v, u, q)$ be the optimal value of the Lagrangian relaxation above. For each $i \in N$ denote by $L_i(v, u, q)$ the optimal value of the following integer linear

program:

$$\min q_i \sum_{t=1}^{T-1} t x_{it}^2 + \sum_{t=0}^{T-1} \sum_{m=1}^{2} v_{tm} \sum_{\tau=\max\{0,t-p_i^m+1\}}^{t} x_{i\tau}^m$$

$$+b(i) \sum_{t=0}^{T-1} u_t \left(\sum_{\tau=0}^{t} x_{i\tau}^1 - \sum_{\tau=0}^{t-b_i} x_{i\tau}^2 \right) \tag{3.3.9}$$

subject to

$$\sum_{t=0}^{T-1} x_{it}^m = 1, \quad \text{for } m \in \{1,2\} \tag{3.3.10}$$

$$\sum_{t=1}^{T-1} t x_{it}^2 - \sum_{t=1}^{T-1} t x_{it}^1 \geq p_i^1 \tag{3.3.11}$$

$$x_{it}^m \in \{0,1\}, \quad \text{for } \ 0 \leq t < T, \text{ and } m \in \{1,2\} \tag{3.3.12}$$

Therefore, for the chosen set of Lagrangian multipliers (v,u,q), $L(v,u,q)$ could be computed as the sum of all $L_i(v,u,q)$ and a linear combination of parameters:

$$L(v,u,q) = \sum_{i=1}^{n} L_i(v,u,q) + \sum_{i=1}^{n} q_i p_i^2 - \sum_{t=0}^{T-1} \sum_{m=1}^{2} v_{tm} - \Omega \sum_{t=0}^{T-1} u_t \tag{3.3.13}$$

Consequently, for the given set (v,u,q), the $L(v,u,q)$ can be found by solving n separate integer problems (3.3.9)-(3.3.12).

Recursive procedure

If job i starts at the time s on the first-stage machine and at the time r - on the second-stage machine, then by virtue of (3.3.10), only $x_{is}^1 = 1$ and $x_{ir}^2 = 1$, hence the value of the objective function (3.3.9) is

$$q_i r + \sum_{t=s}^{s+p_i^1-1} v_{t1} + \sum_{t=r}^{r+p_i^2-1} v_{t2} + b(i) \sum_{t=s}^{r+p_i^2-1} u_t.$$

Define the function $f(r)$ for $p_i^1 \leq r \leq T - p_i^2$ as

$$f(r) = \min_{0 \leq s \leq r-p_i^1} \left(\sum_{t=s}^{s+p_i^1-1} v_{t1} + b(i) \sum_{t=s}^{r+p_i^2-1} u_t \right). \tag{3.3.14}$$

Observe that the initial value of $f(r) = f(p_i^1)$:

$$f(p_i^1) = \sum_{t=0}^{p_i^1-1} v_{t1} + b(i) \sum_{t=0}^{p_i^1+p_i^2-1} u_t,$$

and hence for $r > p_i^1$ the following recursive relation holds:

$$f(r) = \min \left[f(r-1) + b(i)u_{r+p_i^2-1}, \sum_{t=r-p_i^1}^{r-1} v_{t1} + b(i) \sum_{t=r-p_i^1}^{r+p_i^2-1} u_t \right] \tag{3.3.15}$$

Finally, the value of $L_i(v, u, q)$ can be found as

$$L_i(v, u, q) = \min_{p_i^1 \le r \le T-p_i^2} \left(f(r) + q_i r + \sum_{t=r}^{r+p_i^2-1} v_{t2} \right).$$

Description of LR heuristic

The notation similar to that of the previous section will be used to describe the Lagrangian relaxation-based heuristic (hereafter referred to as LR heuristic). Let $LowerBound(v, u, q)$ be the procedure that for the current set of Lagrangian multipliers (v, u, q) calculates the objective function for the Lagrangian relaxation according to (3.3.13). Let π be the permutation of jobs, defined by the Lagrangian relaxation in order of starting times of jobs on the first-stage machine. Let $WAIT(\pi)$ signify the value of the objective function provided by the feasible schedule constructed by the WAIT algorithm with the order π on both machines. Denote by NI_{max} the maximum number of iterations. Let X be the optimal solution vector, obtained during Lagrangian relaxation for the current set of (v, u, q), with coordinates $X_h = x_{it}^m$ for $0 \le h < 2nT$, where $1 \le i \le n$, $0 \le t < T$, $m \in \{1, 2\}$. Denote by A the matrix of the coefficients of the left hand sides of the dualised constraints (3.3.3), (3.3.5) and (3.3.6), and by B the vector of the corresponding right-hand sides. Let λ be a positive coefficient $0 < \lambda \le 2$. The LR heuristic can be summarised as follows:

LR heuristic

1: **Set** $v_{tm} = 0$, $u_t = 0$ for $0 \le t < T$, $m \in \{1, 2\}$; **set** $k = 0$.

2: **Set** $q_1 = 1$ and $q_i = 0$ for $1 < i \le n$.

3: **Set** $BestLB = 0$; $BestUB = T$.
4: **while** $k < NI_{max}$ **do**
5: $LB = LowerBound(v, u, q)$.
6: **if** $LB > BestLB$ **then**
7: $BestLB = LB$.
8: **end if**
9: $UB = WAIT(\pi)$.
10: **if** $UB < BestUB$ **then**
11: $BestUB = UB$.
12: **end if**
13: **Set** $\tau = \lambda \frac{BestUB - LB}{\|AX - B\|^2}$.
14: $(v, u, q) = (v, u, q) + \tau(AX - B)$, $k = k + 1$.
15: **end while**

3.3.3 Bin-packing heuristic

As the name suggests, the bin-packing heuristic utilises the idea of bin-packing. This idea seems to be particularly consonant with "packing" jobs into a buffer in the way allowing to process all the jobs as fast as possible similar to one of the methods of bin-packing [13]. The Bin-packing heuristic partitions all jobs into bins of size Ω, by checking whether a job "fits" into one of the existing bins, and creating a new bin, if the job does not "fit" to any of existing bins. The order of jobs in each bin is determined by Johnson's rule [57], which can be summarized as follows:

- partition N into two sets: $L_1 = \{i \in N : p_i^1 < p_i^2\}$ and $L_2 = \{i \in N : p_i^1 \geq p_i^2\}$;
- first schedule the jobs from L_1 in a non-decreasing order of p_i^1, and
- then schedule the jobs from L_2 in a non-increasing order of p_i^2.

This order provides an optimal makespan for the set of jobs within each bin [57]. Then the bins are ordered in a non-decreasing order of the total buffer requirement

of jobs within each bin. The resulting permutation of all jobs is used by WAIT algorithm with this permutation on both machines. Let $Sort(Priority, Perm)$ be the procedure that sorts a set of elements in a non-decreasing order of elements' priorities $Priority$; the resultant permutation of the elements is recorded in $Perm$. Let $JohnsonSort(Set, Perm)$ be the procedure that sorts a set of jobs $Set \subseteq N$ according to Johnson rule and the resultant permutation is recorded in $Perm$. Let $WAIT(\pi)$ signify the value of the objective function provided by the feasible schedule constructed by WAIT algorithm with the order π on both machines. Denote by $Bin[i] \subseteq N$ a subset of jobs i and let $load_i^1$ and $load_i^2$ signify the sum of p_j^1 and p_j^2, correspondingly, of all jobs $j \in Bin[i]$. Let N_b be the number of "bins", and initially $N_b = 0$. The Bin-packing heuristic can be summarised as follows:

Bin-packing heuristic

Set $N_b = 0$;
for all $i \in N$ **do**
  Set $priority[i] = p_i^1$
end for
$Sort(priority, \pi)$;
for $i = 1$ **to** N **do**
  $allocate = TRUE$;
  if $p_{\pi(i)}^1 > \frac{\Omega}{2}$ **then**
    $Bin[N_b] = \pi(i)$; $load_{N_b}^1 = p_{\pi(i)}^1$; $load_{N_b}^2 = p_{\pi(i)}^2$, $N_b = N_b + 1$;
  else
    while $allocate$ **do**
      Set $j = 0$;
      while $j < N_b$ **do**
        if $load_j^1 + p_{\pi(i)}^1 \leq \Omega$ **and** $load_j^2 + p_{\pi(i)}^2 \leq \Omega$ **then**
          $Bin[j] = Bin[j] \cup \pi(i)$; $load_j^1 = load_j^1 + p_{\pi(i)}^1$; $load_j^2 = load_j^2 + p_{\pi(i)}^2$,
          $j = N_b$; $allocate = FALSE$;
        else
          $j = j + 1$;

end if

end while

if *allocate* **then**

$Bin[N_b] = \pi(i)$; $load^1_{N_b} = p^1_{\pi(i)}$; $load^2_{N_b} = p^2_{\pi(i)}$

$N_b = N_b + 1$; $allocate = FALSE$;

end if

end while

end if

end for

for all $0 \leq i < N_b$ **do**

Set $priority[i] = \frac{1}{load^1_i}$;

end for

$Sort(priority, \pi^{bins})$; **Set** $\pi = \varnothing$, $k = 1$;

for $i = 0$ **to** $N_b - 1$ **do**

$JohnstonSort(Bin[\pi^{bins}(i)], \pi^J_{\pi^{bins}(i)})$;

for $j = 1$ **to** $|Bin[\pi^{bins}(i)]|$ **do**

$\pi(k) = \pi^J_{\pi^{bins}(j)}$; $k = k + 1$;

end for

end for

$WAIT(\pi)$.

3.3.4 Barrier heuristic

The Barrier heuristic utilises the polynomial-time algorithm described in [61]. This algorithm constructs an optimal schedule for a particular case of the considered scheduling problem - when the following condition is satisfied:

$$\max_{i \in N}\{p^1_i, p^2_i\} \leq \frac{\Omega}{5}. \tag{3.3.16}$$

Hence there is a reasonable expectation that the permutation obtained with the help of this algorithm would allow to construct good quality schedules for arbitrary in-

stances of the problem. Below a brief description of the polynomial-time algorithm is provided, followed by a description of the Barrier heuristic.

The polynomial-time algorithm

The polynomial-time algorithm, described in [61] constructs a schedule for an extended set of jobs obtained by adding some auxiliary jobs. To determine the set of the auxiliary jobs, a schedule σ^J is constructed on the set of jobs N ignoring the buffer requirement. The σ^J is a permutation schedule (a permutation schedule is a schedule with the same order of jobs on both machines), constructed according to Johnson's rule, and C^J_{max} is the maximum completion time provided by this schedule.

Let

$$p^1_{max} = \max_{i \in N} p^1_i \quad \text{and} \quad p^2_{max} = \max_{i \in N} p^2_i.$$

Denote by $Idle_1$ and $Idle_2$ the total idle time in σ^J in the interval $[0, C^J_{max}]$ on first and second-stage machines, respectively. Let X and Y be two sets of auxiliary jobs of cardinality

$$|X| = \left\lceil \frac{Idle_2}{p^2_{max}} \right\rceil \quad \text{and} \quad |Y| = \left\lceil \frac{Idle_1}{p^1_{max}} \right\rceil$$

and such that $p^1_i = 0$ and $p^2_i = \frac{Indle2}{|X|}$ for any $i \in X$ and $p^1_i = \frac{Indle1}{|Y|}$ and $p^2_i = 0$ for any $i \in Y$. Observe that for any auxiliary job i $p^1_i \leq p^1_{max}$ and $p^2_i \leq p^2_{max}$. Let σ' be the permutation schedule where the jobs of the set $N' = N \cup X \cup Y$ are scheduled as follows. The first $|X|$ jobs are the jobs constituting X, sequenced in arbitrary order; these jobs are followed by all jobs in N, scheduled according to Johnson's rule; the jobs from N are followed by the arbitrary ordered remaining jobs, i.e. the jobs constituting Y. Let π^J be the permutation of all jobs in N' induced by the order in which these jobs are processed in σ' and π_1 and π_2 be the permutations of the jobs in

$$L'_1 = L_1 \cup X; \quad \text{and} \quad L'_2 = L_2 \cup Y;$$

respectively, induced by π^J. The algorithm below constructs a permutation π which specifies the order in which the jobs are processed in an optimal permutation schedule.

It is convenient to use the following notation:

$$
\begin{aligned}
& n' = n + |X| + |Y|; \\
& l_{k,1}(\pi) = \sum_{j=1}^{k} p^1_{\pi(j)}, \text{ for all } 1 \leq k \leq n'; \\
& l_{k,2}(\pi) = \sum_{j=1}^{k} p^2_{\pi(j)}, \text{ for all } 1 \leq k \leq n'; \\
& R_k(\pi) = l_{k,2}(\pi) - l_{k,1}(\pi), \text{ for all } 1 \leq k \leq n'
\end{aligned}
$$

Let $\pi = \emptyset$ indicate that the permutation π is not specified.

Polynomial(π) algorithm

1: Set $i = 1, i_1 = 1, i_2 = 1, \pi = \emptyset, R_0(\pi) = 0, l_{0,1}(\pi) = 0, l_{0,2}(\pi) = 0$, $H = \frac{2\Omega}{5}$.
2: **while** $i \leq n'$ **do**
3: **if** $R_{i-1}(\pi) < H$ **and** $i_1 \leq |L'_1|$ **then**
4: set $\pi(i) = \pi_1(i_1)$, $i_1 = i_1 + 1$;
5: **else**
6: set $\pi(i) = \pi_2(i_2)$, $i_2 = i_2 + 1$;
7: **end if**
8: set $S^1_{\pi(i)}(\sigma) = l_{i-1,1}(\pi)$ and $S^2_{\pi(i)}(\sigma) = l_{i-1,2}(\pi)$; set $i = i + 1$;
9: **end while**
10: **return** permutation π and schedule σ.

Essentially the Polynomial(π) algorithm constructs the permutation π which ensures that in the resulting schedule the value of $R_k(\pi)$ does not deviate much from the "barrier" H, which allows to observe the buffer capacity Ω. Let $JonhsonCmax(N, C^J_{max})$ be a procedure which for the given set of jobs N constructs a schedule using Johnson's rule and returns the value of its makespan C^J_{max}. As before, let $WAIT(\pi)$ signify the value of the objective function provided by the feasible schedule constructed by WAIT algorithm with the order π on both machines. The Barrier heuristic can be summarised as follows:

Barrier heuristic

1: $JohnstonSort(N, \pi^J)$; $JohnsonCmax(N, C^J_{max})$;

2: **Set** $H = 2\max_{i\in N}\{p_i^1, p_i^2\}$;

3: **Set** $Idle_1 = C_{max}^J - \Sigma_{i\in N} p_i^1$, and $Idle_2 = C_{max}^J - \Sigma_{i\in N} p_i^2$;

4: **Set** $n_x = \left\lceil \frac{Idle_2}{p_{\max}^2} \right\rceil$, $x = \frac{Idle_2}{n_x}$, $X = \emptyset$;

5: **Set** $n_y = \left\lceil \frac{Idle_1}{p_{\max}^1} \right\rceil$, $y = \frac{Idle_1}{n_y}$, $Y = \emptyset$;

6: **for** $i = 1$ **to** n_x **do**

7: **Set** job $i : p_i^1 = 0,\ p_i^2 = x,\ X = X \cup \{i\}$

8: **end for**

9: **for** $i = 0$ **to** $n_y - 1$ **do**

10: **Set** job $i : p_i^1 = y;\ p_i^2 = 0;\ Y = Y \cup \{i\}$

11: **end for**

12: **Set** $L_1 = X$, $L_2 = \emptyset$, $i = 1$;

13: **while** $p^1_{\pi^J(i)} < p^2_{\pi^J(i)}$ **do**

14: $L_1 = L_1 \cup \{\pi^J(i)\}$; $i = i + 1$

15: **end while**

16: **while** $i \leq n$ **do**

17: $L_2 = L_2 \cup \{\pi^J(i)\}$; $i = i + 1$

18: **end while**

19: **Set** $L_2 = L_2 \cup Y$.

20: Run $Polynomial(\pi')$ with sets L_1 and L_2 and barrier H.

21: **Set** $n' = n + |X| + |Y|$, $\pi = \emptyset$; $j = 1$

22: **for** $i = 1$ **to** n' **do**

23: **if** $(\pi'(i) \notin X)$ and $(\pi'(i) \notin Y)$ **then**

24: $\pi(j) = \pi'(i)$, $j = j + 1$

25: **end if**

26: **end for**

27: $WAIT(\pi)$.

3.3.5 Lower Bound

To assess the quality of solutions provided by the described heuristics, for each instance of the considered problem $F2|buffer|C_{max}$ the Johnson's rule could be used to

obtain a lower bound by ignoring the buffer requirement and constructing an optimal schedule. However, if there is a method which explicitly incorporates the size of the buffer Ω, one could expect that this method would calculate better lower bounds. The proposed method to calculate lower bound on the optimal value of the makespan can be described as follows.

Assume that all jobs $i \in N$ are numbered in a non-increasing order of p_i^1. Let $LargeJobs = \{1, 2, ..., k\}$ be the set of the jobs i with buffer requirement $p_i^1 > \frac{\Omega}{2}$. Obviously, no two jobs from this set can be in the buffer together. Hence in any schedule the time, required to process a subset $B_l = \{1, 2, ..., l\} \subseteq LargeJobs$, $1 \leq l \leq k$, is at least $\Sigma_{i=1}^{l}(p_i^1 + p_i^2)$. In addition, for the subset B_l there may exist a subset of smaller jobs $S_l = \{i > l : p_i^1 + p_l^1 > \Omega\}$ such that none of the smaller jobs from S_l can occupy the buffer together with any job from B_l. Note that $N = S_0$ when $LargeJobs = \emptyset$. Let $C(S_l)^J$ be the maximum completion time in a permutation schedule where the jobs from S_l are scheduled according to Johnson's rule and the buffer restriction is ignored. The minimum time required to process sets B_l and S_l is at least

$$\Sigma_{i=1}^{l}\left(p_i^1 + p_i^2\right) + C(S_l)^J.$$

Hence the lower bound LB^{buffer} can be calculated as

$$LB^{buffer} = \max_{1 \leq l \leq k}\left\{\Sigma_{i=1}^{l}\left(p_i^1 + p_i^2\right) + C(S_l)^J\right\} \tag{3.3.17}$$

Denote by $LB^{Johnson} = C(N)^J$. Hence the lower bound LB can be found as

$$\max\{LB^{buffer}, LB^{Johnson}\} \tag{3.3.18}$$

It is easy to see, that if there are no large jobs, then $LB^{buffer} = LB^{Johnson}$.

3.3.6 Computational Experiments

The computational experiments were run for the LR, Bin-packing and Barrier heuristics and aimed to compare their performance against the lower bound (3.3.18).

The test instances were generated randomly with processing times chosen from the interval $[1, 20]$. There were 50 instances in each tested set. In what follows, an instance is described as $n-\Omega_k$, where n is the number of jobs, and Ω_k is the size of the buffer. The experiments were run for sets of instances with 25, 50 and 100 jobs and for buffer sizes $\Omega_1 = p_{max}$, $\Omega_{1.5} = 1.5p_{max}$, $\Omega_{2.5} = 2.5p_{max}$ and $\Omega_{4.5} = 4.5p_{max}$, where $p_{max} = \max\{p^1_{max}, p^2_{max}\}$ is the maximum processing time of an instance. Subgradient algorithm in the LR heuristic was run for 300 iterations; there was a 30 minute time limit per instance for all heuristics.

The Tables 3.13 - 3.16 illustrate the quality of solutions provided by each of the three heuristics: Lagrangian, Bin-packing or Barrier heuristics. Each table represents data for sets of instances with one of the four considered buffer sizes: Ω_1, $\Omega_{1.5}$, $\Omega_{2.5}$ or $\Omega_{4.5}$. For each instance denote by C^1_{max}, C^2_{max} and C^3_{max} the value of the objective function obtained for this instance by LR, Bin-packing or Barrier heuristic correspondingly, and let $Best$ be the smallest of the three values. Then for each C^i_{max}, $i = 1, 2, 3$, the relative difference δ_i is calculated as follows:

$$\delta_i = \frac{C^i_{max} - Best}{Best} \times 100.\%$$

Further, δ_i is attributed to one of the following categories:

- $\delta_i = 0$, hence the heuristic obtained the smallest value of the objective function;
- $0 < \delta_i \leq 2\%$, the heuristic obtained the value of the objective function within 2% from the $Best$;
- $2\% < \delta_i \leq 5\%$, the heuristic obtained the value of the objective function between 2% and 5% from the $Best$;
- $5\% < \delta_i \leq 10\%$, the heuristic obtained the value of the objective function 5% and 10% from the $Best$;
- $\delta_i > 10\%$, the heuristic obtained the value of the objective function exceeding the $Best$ by at least 10%.

Next for each heuristic and for each category above the portion, in %, of the instances in a set with δ_i in the considered category is calculated. Hence each column $2-10$ in the Tables 3.13 - 3.16 represents the proportion of the instances in each of the five categories for sets with 25, 50 or 100 jobs processed by either LR, Bin-packing or Barrier heuristics. The tables show that buffer size and number of jobs affect the performance of the heuristics:

- The LR and Bin-packing heuristics provided better results for instances with smaller buffer sizes Ω_1 and $\Omega_{1.5}$, whereas the Barrier heuristic provided the best solutions for absolute majority of instances with larger buffer $\Omega_{2.5}$ and $\Omega_{4.5}$.
- The LR heuristic obtained solutions within $0-2\%$ of the best for 80-90% of instances with 25 jobs; the Bin-packing heuristic showed best results for $50-\Omega_{1.0}$ and $100-\Omega_{1.0}$ by providing 98-100% of solutions within $0-2\%$ of the best.
- The Barrier heuristic provided best solutions for $96-100\%$ of all instances with larger buffer sizes $\Omega_{2.5}$ and $\Omega_{4.5}$, with only exception of $25-\Omega_{2.5}$ instances, for which the LR heuristic provided solutions within $0-2\%$ of the best for 78% of the instances.

Table 3.13: Quality of solution for instances with buffer size $\Omega_{1.0}$, proportion of instances, in %

	25 jobs			50jobs			100 jobs		
	LR	Bin	Barrier	LR	Bin	Barrier	LR	Bin	Barrier
Best, $\delta_i = 0$	76	30	2	20	80	2	2	98	0
$0\% \leq \delta_i \leq 2\%$	14	22	2	28	8	2	4	2	4
$2\% \leq \delta_i \leq 5\%$	10	38	10	40	10	18	36	0	32
$5\% \leq \delta_i \leq 10\%$	0	10	48	12	2	62	58	0	56
$\delta_i > 10\%$	0	0	38	0	0	16	0	0	8

The box-plot charts on Figures 3-6 - 3-8 represent the the relative error for instances with 25, 50 and 100 jobs correspondingly, where the relative error RE_i is calculated for each C^i_{max} as

$$RE_i = \frac{C^i_{max} - LB}{LB} \times 100\%,$$

Table 3.14: Quality of solution for instances with buffer size $\Omega_{1.5}$, proportion of instances, in %

	25 jobs			50jobs			100 jobs		
	LR	Bin	Barrier	LR	Bin	Barrier	LR	Bin	Barrier
Best, $\delta_i = 0$	74	24	6	56	32	14	22	64	14
$0\% \leq \delta_i \leq 2\%$	8	14	2	18	26	14	8	26	32
$2\% \leq \delta_i \leq 5\%$	10	10	12	20	14	30	46	10	20
$5\% \leq \delta_i \leq 10\%$	4	32	30	6	22	34	20	0	22
$\delta_i > 10\%$	4	20	50	0	6	8	4	0	12

Table 3.15: Quality of solution for instances with buffer size $\Omega_{2.5}$, proportion of instances, in %

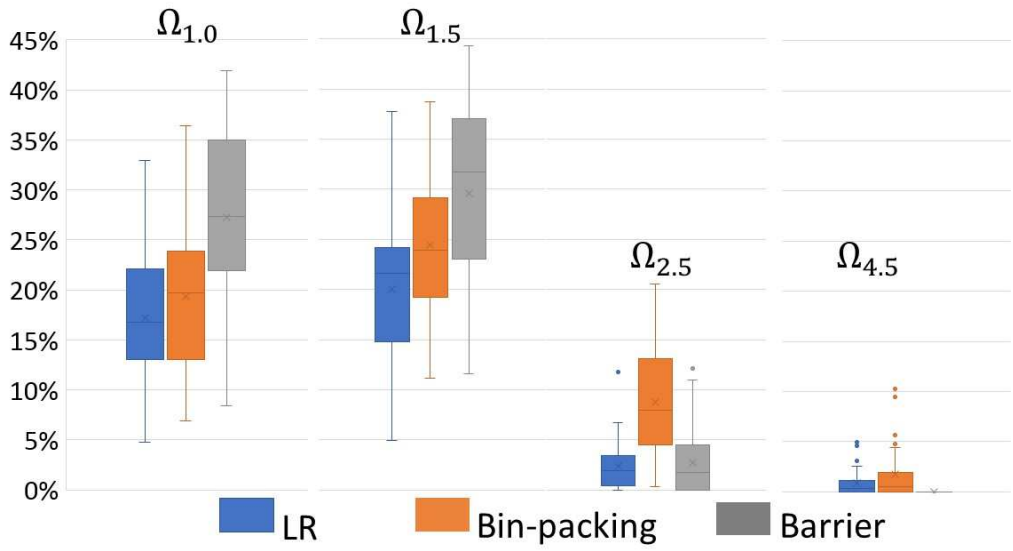
	25 jobs			50jobs			100 jobs		
	LR	Bin	Barrier	LR	Bin	Barrier	LR	Bin	Barrier
Best, $\delta_i = 0$	56	0	58	18	0	88	0	0	100
$0\% \leq \delta_i \leq 2\%$	30	18	20	24	0	10	2	0	0
$2\% \leq \delta_i \leq 5\%$	14	22	18	46	2	2	52	0	0
$5\% \leq \delta_i \leq 10\%$	0	30	4	12	32	0	46	2	0
$\delta_i > 10\%$	0	30	0	0	66	0	0	98	0

Table 3.16: Quality of solution for instances with buffer size $\Omega_{4.5}$, proportion of instances, in %

	25 jobs			50jobs			100 jobs		
	LR	Bin	Barrier	LR	Bin	Barrier	LR	Bin	Barrier
Best, $\delta_i = 0$	48	26	100	20	8	100	12	0	100
$0\% \leq \delta_i \leq 2\%$	36	52	0	60	4	0	52	0	0
$2\% \leq \delta_i \leq 5\%$	16	12	0	18	20	0	34	16	0
$5\% \leq \delta_i \leq 10\%$	0	6	0	2	30	0	2	84	0
$\delta_i > 10\%$	0	4	0	0	38	0	0	0	0

where lower bound LB is calculated according to (3.3.18).

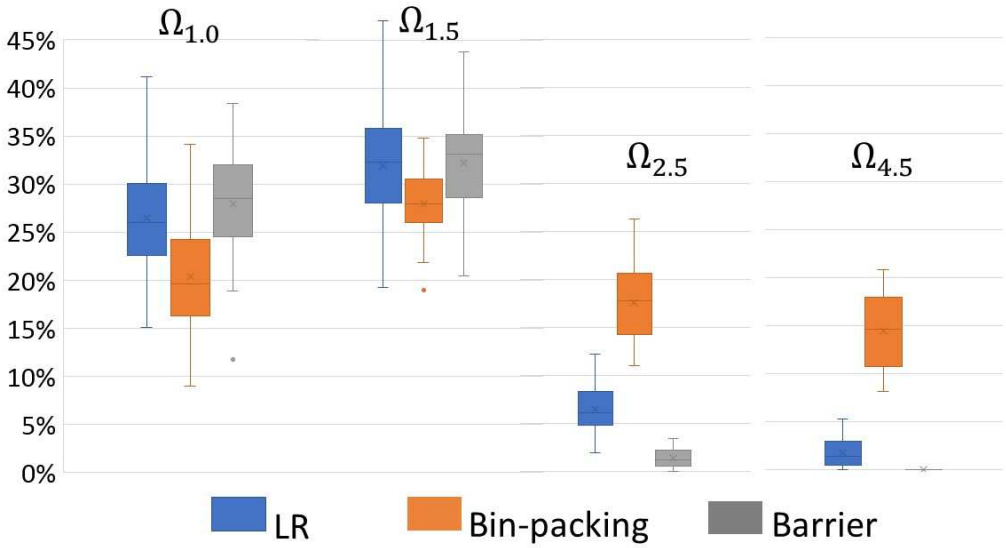
Figure 3-6: Relative Error for 25 jobs instances



For smaller buffer sizes $\Omega_{1.0}$ and $\Omega_{1.5}$ the LR and the Bin-packing heuristics have smaller relative errors then the Barrier heuristic for most of the instances with 25 and 50 jobs, however for instances with 100 jobs Bin-packing heuristic provided tighter solutions than the Barrier and LR heuristics. For larger buffer sizes $\Omega_{2.5}$ and $\Omega_{4.5}$ the Barrier heuristic had the tightest solutions - for instances with $\Omega_{2.5}$ all relative errors were within $5-10\%$, and for the buffer size $\Omega_{4.5}$ the heuristic provided optimal solutions for all instances (with $RE = 0\%$). The LR heuristic had solutions with relative errors within $5-10\%$ for the larger buffer sizes, however the Bin-packing heuristic provided solutions within larger relative errors of $10-20\%$ for most of instances with the larger buffer sizes $\Omega_{2.5}$ and $\Omega_{4.5}$.

The average time spent by each heuristic per instance is presented on the Figure 3-9. Both Barrier and Bin-packing heuristics were fast and spent $0.000041-0.00068$ seconds per instance, while the LR heuristic was significantly slower and its CPU time varied between $1.6-7.2$ seconds per 25 jobs instance, $10.3-48.3$ seconds per 50 jobs instance and $1.5-6$ minutes per 100 jobs instance. In comparison, when ten $25-\Omega_{4.5}$ instances were tested by running a straightforward integer program (CPLEX), it took

Figure 3-7: Relative Error for 50 jobs instances



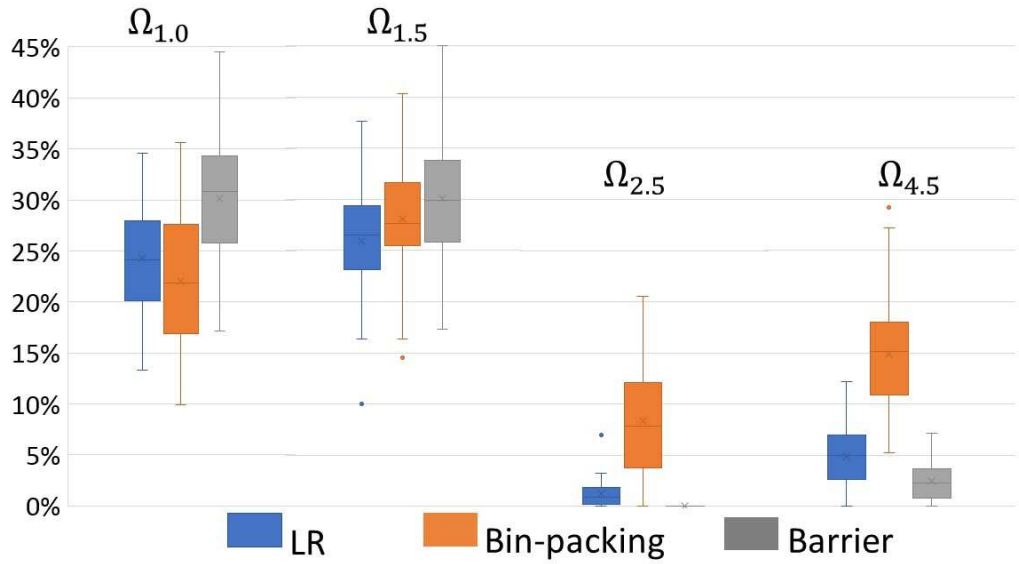
30 minutes for each instance to only determine that an integer solution exists, and for every instance the best "exit" value of the objective function was greater than the corresponding values obtained by the heuristics. It is reasonable to assume that the Barrier and Bin-packing heuristics were faster due to two factors: firstly, due to taking into account the buffer size explicitly while deriving the permutation of jobs, and secondly, due to less computational effort required to obtain this permutation than that of LR heuristic.

The comparison of LB^{buffer} vs $LB^{Johnson}$ is represented on the Figure 3-10. For each instance the δ was calculated as:

$$\delta = \frac{LB^{buffer} - LB^{Johnson}}{LB^{Johnson}} \times 100\%$$

The box plots of the δs were constructed for instances with 25, 50 and 100 jobs and buffer sizes $\Omega_{1.0}$ and $\Omega_{1.5}$. Please note that for larger buffer sizes $\Omega_{2.5}$ and $\Omega_{4.5}$ $LB^{buffer} = LB^{Johnson}$. It is clear that for the instances with the buffer size $\Omega_{1.0}$ LB^{buffer} was better (greater) than lower bound the $LB^{Johnson}$ by $5\% - 57\%$ across all instances with a median improvement $\delta = 29\%$. However, for the buffer size $\Omega_{1.5}$

Figure 3-8: Relative Error for 100 jobs instances



LB^{buffer} was smaller than $LB^{Johnson}$ for absolute majority of instances by median $\delta = 20\%$.

In summary, the computational experiments demonstrated that all three heuristics provide good feasible solutions for the considered scheduling problem. Further, the Barrier heuristic is a fast algorithm that generates near-optimal/optimal schedules for the instances with a larger buffer size. The LR and Bin-packing heuristics provide tighter results for the instances with smaller buffer and 25-50 jobs. The proposed method to calculate the lower bound considerably tightens the relative error for instances with buffer size $\Omega_{1.0}$. All three heuristics outperform the straightforward integer programming approach (CPLEX).

3.3.7 Conclusion

In this section, Lagrangian relaxation and decomposition-based approach, developed in the previous section was applied to a different flow shop problem with a job-dependent buffer: the problem of minimisation of the maximum completion for the two-stage flow shop with a a job-dependent buffer. Two other heuristics are also presented: Bin-packing and Barrier heuristics. The three heuristics are fast algorithms

Figure 3-9: Average time per instance

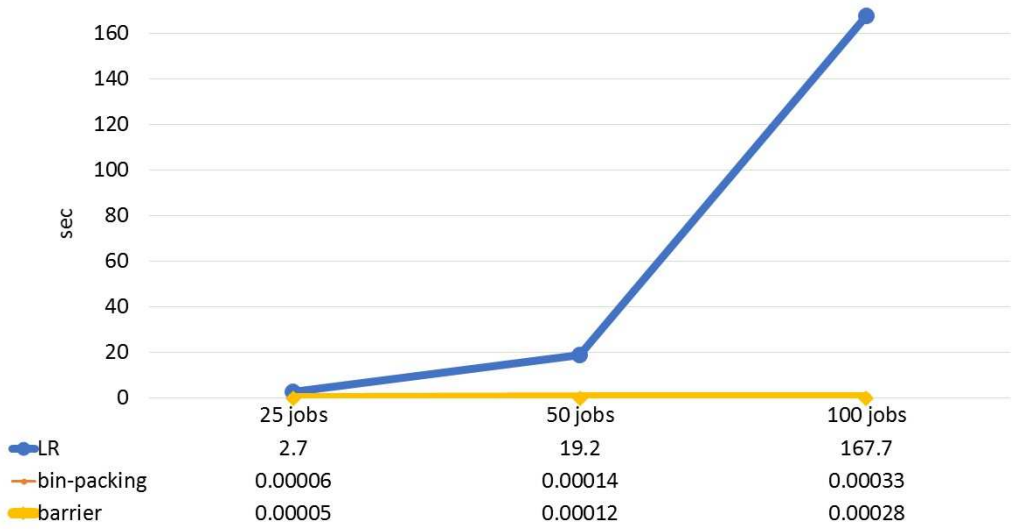
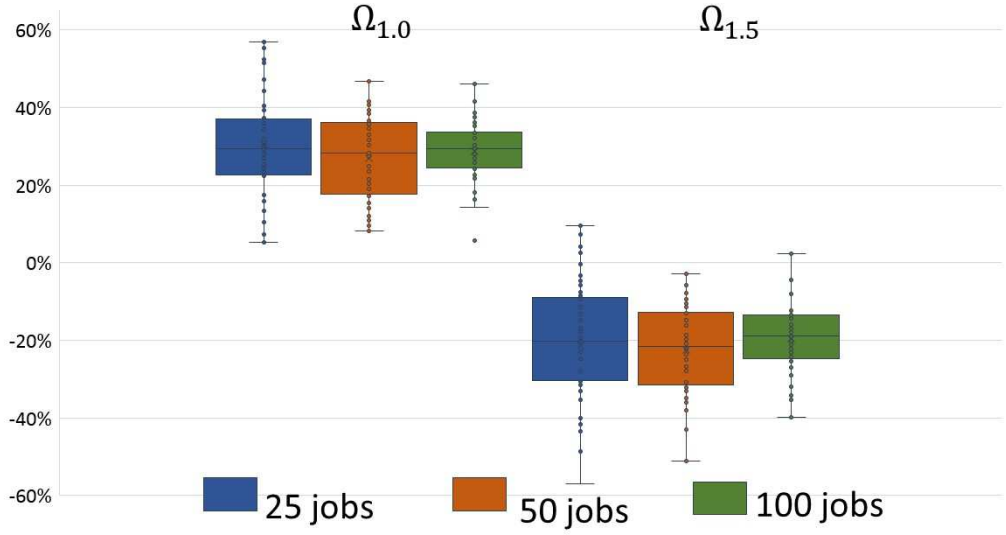


Figure 3-10: LB^{buffer} vs $LB^{Johnson}$



that provide good feasible solutions for the considered problem. All heuristics' results surpass the straightforward application of CPLEX optimisation software.

3.4 Two-stage hybrid flow shop with a storage and objective to minimise total weighted tardiness

3.4.1 Problem Description

In what follows, the Lagrangian relaxation and decomposition are applied to the problem that is motivated by various systems where the change of the means of transportation requires considerable storage space (a buffer). For example, in the case of supply chains of mineral resources, the material is transported by a fleet of trains or trucks and is stored in a stockpile on the so-called pads before being loaded for the second stage of transportation. At this second stage the stockpiles are transported in groups. It is common to reserve the entire space for all stockpiles of a group before the arrival of the first load for these stockpiles and to release this space only after the completion of loading all stockpiles of the group for further transportation [24]. There can be several loaders involved in the uploading the groups of stockpiles. The problem can be described as follows.

The set of jobs $N = \{1, ..., n\}$ is to be processed on a two-stage flow shop; all jobs are partitioned into n_b predefined batches; batch k is comprised of a set of jobs N_k, $1 \leq k \leq n_b$; m_1 and m_2 are the numbers of parallel identical machines on the first and the second stage, correspondingly. Let p_i be the processing time of job i on a first-stage machine and ρ_k be the processing time of batch k. All processing times are integer. Each batch k has an associated due date d_k and a weight w_k and requires $b(k)$ units of the buffer capacity which it seizes from the start of processing of the earliest job of the batch on a first-stage machine till the batch's completion on a second-stage machine. At any time, the total buffer requirement cannot exceed the buffer capacity Ω. It is assumed that the processing of jobs commences at time $t = 0$. A schedule is the two sets, $\{S_1^1, ..., S_n^1\}$ and $\{S_1^2, ..., S_{n_b}^2\}$, where S_i^1 is the time when a first-stage machine starts processing job i and S_k^2 is the time when a second-stage machine starts processing batch k. The objective is to minimise the total weighted tardiness $\sum_{k=1}^{n_b} w_k T_k$, where $C_k = S_k^2 + \rho_k$ is the completion time of batch

k and $T_k = \max\{0, C_k - d_k\}$ is the tardiness of batch k. The considered problem is NP-hard in a strong sense, as even a particular case of the problem, when each stage constitutes of only one machine, all weights are equal to one, and all due dates are equal to zero and there is no buffer, is NP-hard in a strong sense [31].

3.4.2 Choice of the Planning Horizon

Let T be the planning horizon, i.e. $C_k \leq T$ for all batches. A choice of T has a significant impact on the efficiency of the solution method such as Lagrangian relaxation. One obvious value for T can be estimated as

$$T = \sum_{i \in N} p_i + \sum_{1 \leq k \leq n_b} \rho_k. \tag{3.4.1}$$

However, this upper bound of the planning horizon can be tightened which is beneficial to the performance of the Lagrangian relaxation solution approach.

Let σ^* be an optimal schedule for criterion of the total weighted tardiness. It is assumed that σ^* is an active schedule.

Definition 3 *A time interval is incomplete, if during the entire interval at least one machine is idle on the first-stage at any point of time, and there are no batches processed on the second stage.*

Definition 4 *A time interval is full, if during the entire interval there are no idle machines on the first-stage at any point of time, and there are no batches processed on the second stage.*

Definition 5 *A time interval is loaded, if during the entire interval there is a batch processed on the second stage at any point of time.*

Lemma 3 *For any incomplete time interval there is at least one machine on the first stage which is idle during the entire time interval.*

Proof: Consider the machine on the first stage which is idle at the start of the interval. If any job starts on this machine during the interval, it would imply that

either the schedule is not active and the job could start earlier, or that there was not enough space in the buffer before the start of the job, hence a batch had to be released from the buffer immediately before the job started. This would imply that the batch has been processed during the interval, which contradicts to the interval being incomplete.□

Definition 6 *Let the list of batches be constructed in non-decreasing order of batches' completion times. An incomplete time interval is canonically partitioned, if it is partitioned into subintervals such that for each subinterval there is a job selected as follows. Among all jobs which are processed during the entire subinterval, select a job from the batch on the earliest position on the list of batches. The selected job is a canonical cover of the subinterval.*

Theorem 1 *The set of canonical covers of all incomplete intervals contains at most one job from each batch.*

Proof: Assume that for the incomplete intervals $[t_1, t_2]$ and $[t_3, t_4]$, $t_2 \leq t_3$, the corresponding canonical covers are jobs i_1 and i_2, and both jobs are from the same batch j and $S^1_{i_2} > t_1$. If the intervals are consecutive, i.e. $t_2 = t_3$, then by virtue of Lemma 3, there is an idle machine on the first stage for the entire time interval $[t_1, t_2]$, hence i_2 could start earlier, which contradicts to the schedule being active.

Assume that the intervals are not consecutive, i.e $t_2 < t_3$. Then there is at least one machine from either stage that is not idle on the interval $[t_2, t_3]$, overwise the schedule would not be active. Assume that a batch k starts on second stage at time S^2_k, and $t_2 \leq S^2_k < t_3$. There are two possibilities: either the second-stage machine is idle before batch k starts and there is a job h from the batch k such that the completion time of the job $C_h = S^2_k$ and the starting time of job h $S^1_h > t_2$; or there are batches scheduled on the second stage before k which prevent k to start earlier. Observe, that since both intervals are incomplete, the batches will have to start and complete during the interval $[t_2, t_3]$.

In the former case, if $S^1_h \leq t_1$, then h had to be selected as the canonical cover for the interval $[t_1, t_2]$, as batch k completes before batch j. If $S^1_h > t_1$, then by virtue of

Lemma 3 there is an idle machine at time t_1 on the first stage, hence h could have started earlier, which contradicts to the schedule being active. In the latter case, select the batch which started before k with the earliest starting time such that the second-stage machine is idle before the batch starts on the second stage and repeat the reasoning for the former case.

Since both intervals are incomplete, there is no batch processed on the entire interval $[t_1, t_4]$. Hence there is at least one machine is not idle on the first stage during the interval $[t_2, t_3]$. If there is a full interval within $[t_2, t_3]$, then some batch must have completed on the second stage to release buffer space immediately before the full interval started, overwise by virtue of Lemma 3 some job scheduled during the full interval could have started earlier on an idle machine. However, it has been shown above that no batch is processed during $[t_1, t_4]$. Hence the only option left to consider is that $[t_2, t_3]$ is an incomplete interval. Similar to the above, in this case by virtue of Lemma 3 there is an idle machine on the first stage on both intervals $[t_1, t_2]$ and $[t_2, t_3]$, and hence i_2 could have started earlier, which contradicts to the schedule being active. □

Let I,F and L be the total length of incomplete, full an loaded time intervals, correspondingly, in the schedule σ^*. Then taking into account the Theorem 1,

$$\begin{aligned}
&\max_{1\le k\le n_b} C_k(\sigma^*) = L + F + I \\
&\le \sum_{1\le k\le n_b} \rho_k + \frac{\sum_{i\in N} p_i - I}{M} + I = \sum_{1\le k\le n_b} \rho_k + \frac{\sum_{i\in N} p_i}{M} + (1 - \frac{1}{M})I \\
&\le \sum_{1\le k\le n_b} \rho_k + \frac{\sum_{i\in N} p_i}{M} + (1 - \frac{1}{M}) \sum_{1\le k\le n_b} \max_{i\in N_k} p_i
\end{aligned} \tag{3.4.2}$$

3.4.3 Integer Programming Formulation

By virtue of (3.4.2), set the value of the planning horizon as following:

$$T = \sum_{1\le k\le n_b} \rho_k + \frac{\sum_{i\in N} p_i}{m_1} + (1 - \frac{1}{m_1}) \sum_{1\le k\le n_b} \max_{i\in N_k} p_i \tag{3.4.3}$$

For a job $i \in N$ and an integer index $t \in [0, T)$, define

$$x_{it} = \begin{cases} 1, & \text{if } S_i^1 = t \\ 0, & \text{otherwise.} \end{cases}$$

For a batch k, $1 \leq k \leq n_b$, and an integer index $t \in [0, T)$, define

$$y_{kt} = \begin{cases} 1, & \text{if } S_k^2 = t \\ 0, & \text{otherwise.} \end{cases}$$

For a batch k, $1 \leq k \leq n_b$, and an integer index $t \in [0, T)$, define

$$z_{kt} = \begin{cases} 1, & \text{if } \sum\limits_{i \in N_k} \sum\limits_{\tau=0}^{t} x_{i\tau} > 0 \\ 0, & \text{otherwise.} \end{cases}$$

Hence if there is $i \in N_k$ such that $S_i^1 \leq t$, then $z_{kt} = 1$. The considered scheduling problem can be formulated as the following integer linear program:

$$\min \sum_{k=1}^{n_b} w_k T_k, \tag{3.4.4}$$

subject to

$$\sum_{t=0}^{T-1} x_{it} = 1, \text{ for } 1 \le i \le n \tag{3.4.5}$$

$$\sum_{i=1}^{n} \sum_{\tau=\max\{0,t-p_i+1\}}^{t} x_{i\tau} \le m_1, \text{ for } 0 \le t < T \tag{3.4.6}$$

$$\sum_{i\in N_k} \sum_{\tau=0}^{t} x_{i\tau} - z_{kt}|N_k| \le 0, \text{ for } 0 \le t < T,\ 0 \le k \le n_b \tag{3.4.7}$$

$$\sum_{i\in N_k} \sum_{\tau=0}^{t} x_{i\tau} - z_{kt} \ge 0, \text{ for } 0 \le t < T,\ 0 \le k \le n_b \tag{3.4.8}$$

$$\sum_{t=0}^{T-1} t(y_{kt} - x_{it}) \ge p_i, \text{ for } i \in N_k,\ 0 \le k \le n_b \tag{3.4.9}$$

$$\sum_{k=1}^{n_b} b(k) \left(z_{kt} - \sum_{\tau=0}^{t-\rho_k} y_{k\tau} \right) \le \Omega, \text{ for } 0 \le t < T \tag{3.4.10}$$

$$\sum_{k=1}^{n_b} \sum_{\tau=\max\{0,t-\rho_k+1\}}^{t} y_{k\tau} \le m_2, \text{ for } 0 \le t < T \tag{3.4.11}$$

$$\sum_{t=0}^{T-1} y_{kt} = 1, \text{ for } 1 \le k \le n_b \tag{3.4.12}$$

$$T_k \ge \sum_{t=0}^{T-1} t y_{kt} + \rho_k - d_k \text{ and } T_k \ge 0, \text{ for } 1 \le k \le n_b \tag{3.4.13}$$

$$x_{it},\ y_{kt},\ z_{kt} \in \{0,1\}, \text{ for } i \in N, 0 \le t < T, 1 \le k \le n_b \tag{3.4.14}$$

The constraints can be summarised as follows: (3.4.5) and (3.4.12) guarantee that a job or a batch starts only once on the first or on the second stage, correspondingly; (3.4.7) and (3.4.8) define the value of z_{kt}; (3.4.6), (3.4.10) and (3.4.11) are capacity constraints for the first, the second stages and the buffer, correspondingly; (3.4.9) guarantees that the job cannot start processing on the second stage till the job is completed on the first stage; (3.4.13) defines tardiness T_k.

3.4.4 Lagrangian relaxation and decomposition

To obtain the Lagrangian relaxation the constraints (3.4.6), (3.4.10) and (3.4.11) are dualised:

$$\begin{aligned}
&\min \sum_{k=1}^{n_b} w_k T_k + \sum_{t=0}^{T-1} v_t \left(\sum_{i=1}^{n} \sum_{\tau=\max\{0,t-p_i+1\}}^{t} x_{i\tau} - m_1 \right) \\
&+ \sum_{t=0}^{T-1} u_t \left(\sum_{k=1}^{n_b} b(k) \left(z_{kt} - \sum_{\tau=0}^{t-\rho_k} y_{k\tau} \right) - \Omega \right) \\
&+ \sum_{t=0}^{T-1} q_t \left(\sum_{k=1}^{n_b} \sum_{\tau=\max\{0,t-\rho_k+1\}}^{t} y_{k\tau} - m_2 \right), \qquad (3.4.15)
\end{aligned}$$

where $v_t,\ u_t,\ q_t,\ 0 \le t < T$, are nonnegative Lagrangian multipliers, subject to (3.4.5), (3.4.7)-(3.4.9), (3.4.12)-(3.4.14). This problem can be decomposed into n_b subproblems as follows. Let (v, u, q) be the sets of all v_t, u_t and q_t. For each batch k, $1 \le k \le n_b$, and chosen (v, u, q) define the following linear integer program:

$$\begin{aligned}
&\min w_k T_k + \sum_{i \in N_k} \sum_{t=0}^{T-1} v_t \sum_{\tau=\max\{0,t-p_i+1\}}^{\tau} x_{it} \\
&+ b(k) \sum_{t=0}^{T-1} u_t \left(z_{kt} - \sum_{\tau=0}^{t-\rho_k} y_{k\tau} \right) + \sum_{t=0}^{T-1} q_t \sum_{\tau=\max\{0,t-\rho_k+1\}}^{t} y_{k\tau} \qquad (3.4.16)
\end{aligned}$$

subject to

$$\sum_{t=0}^{T-1} x_{it} = 1, \ \text{ for } i \in N_k \qquad (3.4.17)$$

$$\sum_{i \in N_k} \sum_{\tau=0}^{t} x_{i\tau} - z_{kt}|N_k| \le 0, \ \text{ for } 0 \le t < T \qquad (3.4.18)$$

$$\sum_{i \in N_k} \sum_{\tau=0}^{t} x_{i\tau} - z_{kt} \ge 0, \ \text{ for } 0 \le t < T \qquad (3.4.19)$$

$$\sum_{t=0}^{T-1} t(y_{kt} - x_{it}) \ge p_i, \ \text{ for } i \in N_k \qquad (3.4.20)$$

$$\sum_{t=0}^{T-1} y_{kt} = 1 \qquad (3.4.21)$$

$$T_k \ge \sum_{t=0}^{T-1} t y_{kt} + \rho_k - d_k \text{ and } T_k \ge 0, \qquad (3.4.22)$$

$$x_{it},\ y_{kt},\ z_{kt} \in \{0, 1\}, \ \text{ for } i \in N_k, 0 \le t < T \qquad (3.4.23)$$

Then the optimal value $LR(v,u,q)$ of the objective function (3.4.15) can be expressed as the sum of the optimal values $Z_k(v,u,q)$ of the objective functions (3.4.16) for $1 \le k \le n_b$ and the linear combination of the parameters m_1, m_2, Ω and the chosen Lagrangian multipliers:

$$LR(v,u,q) = \sum_{k=1}^{n_b} Z_k(v,u,q) - \sum_{t=0}^{T-1} (v_t m_1 + u_t \Omega + q_t m_2) \tag{3.4.24}$$

To solve the Lagrangian relaxation for the chosen Lagrangian multipliers it is sufficient to solve n_b separate integer linear programs (3.4.16) - (3.4.23). Each of the n_b subproblems is solved by the dynamic programming procedure described below.

Recursive procedure

Assume that a batch k starts on the first stage at time s and it is completed on the second stage at time $t + \rho_k$. Hence the batch k must be completed on the first stage by the time t. Then (3.4.16) can be presented as

$$\begin{aligned} &\min w_k T_k + \sum_{i \in N_k} \sum_{\tau=s}^{t-1} v_\tau \sum_{\lambda=\max\{s,\tau-p_i+1\}}^{\tau} x_{i\lambda} \\ &+b(k) \sum_{\tau=s}^{t+\rho_k-1} u_\tau + \sum_{\tau=t}^{t+\rho_k-1} q_\tau \sum_{\lambda=\max\{t,\tau-\rho_k+1\}}^{\tau} y_{k\lambda} \end{aligned} \tag{3.4.25}$$

Define $g(s,t)$ and $f(s,t)$ as follows:

$$\begin{aligned} g(s,t) &= w_k T_k + b(k) \sum_{\tau=s}^{t+\rho_k-1} u_\tau + \sum_{\tau=t}^{t+\rho_k-1} q_\tau, \\ f(s,t) &= \min \sum_{i \in N_k} \sum_{\tau=s}^{t-1} v_\tau \sum_{\lambda=\max\{s,\tau-p_i+1\}}^{\tau} x_{i\lambda}. \end{aligned}$$

Taking into account that $t - s \ge p_i$, for each job $i \in N_k$ define $f_i(s,t)$ as follows:

$$f_i(s,t) = \min_{x_{i\tau}=1;\tau=s,\ldots,t-p_i} \sum_{\lambda=s}^{t-1} v_\lambda \sum_{\alpha=\max\{0,\lambda-p_i+1\}}^{\lambda} x_{i\alpha}.$$

Since $\sum_{\alpha=\max\{0,\lambda-p_i+1\}}^{\lambda} x_{i\alpha} = 1$ only for $\tau \le \lambda \le \tau + p_i - 1$,

$$f_i(s,t) = \min_{\tau=s,\dots,t-p_i} \sum_{\lambda=\tau}^{\tau+p_i-1} v_\lambda.$$

For a fixed s the initial value of $f_i(s,t)$ is

$$f_i(s, s+p_i) = \sum_{\lambda=s}^{s+p_i-1} v_\lambda.$$

Further,

$$f_i(s,t+1) = \min\{f_i(s,t), \sum_{\lambda=t-p_i+1}^{t} v_\lambda.\}$$

Hence

$$f(s,t) = \min_{i\in N_k} \left\{ \sum_{\lambda=s}^{s+p_i-1} v_\lambda + \sum_{j\neq i; j\in N_k} f_j(s,t) \right\};$$

$$Z_k(v,u,q) = \min_{[s,t]} \left(f(s,t) + g(s,t)\right), \quad 0 \le s < s+\rho_k \le t \le T - \rho_k.$$

3.4.5 Lagrangian relaxation-based optimisation procedure

To describe the Lagrangian relaxation-based optimisation procedure (hereafter referred to as LR procedure) the notation similar to that of the previous sections is used. Let $LowerBound(v,u,q)$ be a procedure which for the current set of Lagrangian multipliers (v,u,q) calculates the objective function for the Lagrangian relaxation according to (3.4.24). Let π be the permutation of batches, defined by the Lagrangian relaxation in order of starting times of batches on the first stage, and π_k be the permutation of jobs in a batch k, defined by the Lagrangian relaxation in order of starting times of the jobs on the first or on the second stage. Let $WAIT(\pi, \pi_1, ..., \pi_k)$ signify the value of the objective function provided by the feasible schedule constructed by the following modification of the WAIT algorithm: batches are scheduled with the order π on both stages, and jobs of each batch k are scheduled in order π_k on the first

stage. Denote by NI_{max} the maximum number of iterations. Let X be the optimal solution vector, obtained during Lagrangian relaxation for the current set of (v, u, q), with coordinates $X_h = x_{it}$ for $0 \leq h < nT$ and $X_h = y_{kt}$ for $nT \leq h < kT + nT$, where $1 \leq i \leq n$, $0 \leq t < T$, $1 \leq k \leq n_b$ and $0 \leq t < T$. Denote by A the matrix of the coefficients of the left hand sides of the dualised constraints (3.4.6), (3.4.10) and (3.4.11), and by B the vector of the corresponding right hand sides. Let λ be a positive coefficient $0 < \lambda \leq 2$. The LR procedure can be summarised as follows:

LR Procedure

Set $v_t = 0$, $u_t = 0$ and $q_t = 0$ for $0 \leq t < T$, $m \in \{1, 2\}$; **set** $k = 0$.
Set $BestLB = 0$; $BestUB = T$.
while $k < NI_{max}$ **do**
  $LB = LowerBound(v, u, q)$.
  if $LB > BestLB$ **then**
    $BestLB = LB$.
  end if
  $UB = WAIT(\pi, \pi_1, ..., \pi_k)$.
  if $UB < BestUB$ **then**
    $BestUB = UB$.
  end if
  Set $\tau = \lambda \frac{BestUB - LB}{\|AX - B\|^2}$.
  $(v, u, q) = (v, u, q) + \tau(AX - B)$, $k = k + 1$.
end while

3.4.6 Scaling

In this subsection we will look at the vector $AX - B$ introduced above in more detail. Denote by a_i, $0 \leq i < 3T$, the coordinates of the current vector $AX - B$. Observe that the coordinates corresponding to the buffer capacity constraint (3.4.10) have

indices $T \leq i < 2T$. Then $\|AX - B\|^2$ can be presented as

$$\|AX - B\|^2 = \underbrace{\sum_{i=0}^{T-1} a_i^2 + \sum_{i=2T}^{3T-1} a_i^2}_{machines' \ constarints} + \underbrace{\sum_{i=T}^{2T-1} a_i^2}_{buffer \ constarints} \tag{3.4.26}$$

It was observed in the initial computational experiments that if the size of the buffer requirements is significantly larger than the size of other requirements (say, the number of machines on each stage), then the magnitude of the coordinates a_i, corresponding to the buffer constraints, is several times greater than the magnitude of the other coordinates. As a result of the disproportion, the step τ, defined in step 13 of the LR procedure, converges to zero after just a few iterations, which in turn does not allow to update the Lagrangian multipliers efficiently. To improve the convergence of the algorithm, the coordinates of $AX - B$, which correspond to the buffer capacity constraints, can be scaled by replacing the buffer capacity Ω by $\tilde{\Omega} = \alpha\Omega$ and by replacing the buffer requirement $b(k)$ by $b(\tilde{k}) = \alpha b(k)$ for each batch k, $1 \leq k \leq n_b$, where $\alpha > 0$ is a scaling coefficient. Observe that the integer program with the scaled buffer requirements and buffer capacity is equivalent to the original integer program. The coefficient α can be either a chosen constant, for example $\alpha = 0.0001$, or it can be adjusted dynamically at each iteration as follows:

$$\alpha = \sqrt{\frac{\sum_{i=0}^{T-1} a_i^2 + \sum_{i=2T}^{3T-1} a_i^2}{\sum_{i=T}^{2T-1} a_i^2}}$$

Then the squared magnitude of the vector with adjusted coordinates $\|\tilde{A}X - \tilde{\Omega}\|^2$ is

$$\|\tilde{A}X - \tilde{\Omega}\|^2 = \sum_{i=0}^{T-1} a_i^2 + \sum_{i=2T}^{3T-1} a_i^2 + \alpha_j^2 \sum_{i=T}^{2T-1} a_i^2 = 2\left(\sum_{i=0}^{T-1} a_i^2 + \sum_{i=2T}^{3T-1} a_i^2\right).$$

Thus the magnitude of the larger coordinates does not interfere with τ's size, τ converges to zero more gradually, which in turn improves the convergence of the LR procedure.

3.4.7 Permutation heuristic

Theoretically, the optimal value of (3.4.4) can be found by the complete enumeration of all possible permutations of batches and jobs. However, it is infeasible to even generate all possible permutations for more than twenty elements (see, for example, [93] - [92]). Instead, in the proposed permutation heuristic a certain order of the jobs within each batch is selected without enumerating all possible permutations of the jobs; then only permutations of batches are enumerated; finally, the batches are scheduled on the first and the second stages in the same order.

It has been shown in [37] that for a set of independent jobs, scheduled by a list algorithm on m parallel machines with the LPT (*Longest Processing Time first*) rule, the maximum completion time of the set C_{max}^{LPT} is bounded by the following tight worst-case performance guarantee

$$\frac{C_{max}^{LPT}}{C_{max}^{*}} \leq \frac{4}{3} - \frac{1}{3m},$$

where C_{max}^{*} is the optimal value of the makespan criterion. The permutation heuristic schedules jobs of each batch on the first stage parallel machines with the LPT rule; and it enumerates all possible permutations of the given n_b batches and schedules batches with the same order on each stage; then the schedule with the best value of the objective function is chosen. The initial computational experiments showed that this approach is very fast for small instances of 25-50 jobs with 3-5 batches and it produces values within 5% of the optimal value delivered by CPLEX for about 90% of the instances. Denote by $N_p = n_b!$ the total number of permutations of n_b batches, and let π^i be the current permutation of batches, and π_k be the permutation of jobs in a batch k, defined by the LPT rule. Let $WAIT(\pi^i, \pi_1, ..., \pi_k)$ signify the value of the objective function provided by the feasible schedule constructed by the following modification of the WAIT algorithm: batches are scheduled with the order π^i on both stages, and jobs of each batch k are scheduled in order π_k on the first stage. The permutation heuristic can be summarised as follows:

Permutation heuristic

1: **Set** $BestUB = T$.
2: **for** $i = 1$ **to** N_p **do**
3: $UB = WAIT(\pi^i, \pi_1, ..., \pi_k)$.
4: **if** $UB < BestUB$ **then**
5: $BestUB = UB$.
6: **end if**
7: **end for**

3.4.8 Computational experiments

The test instances were generated randomly with the processing times chosen from the interval $[1, 10]$, the weights chosen from the interval $(0, 2]$ and the buffer requirements chosen from the interval $[100, 1000]$. A due date for each batch k was chosen from the interval $[\rho_k + 10, 2(\rho_k + 10)]$. In what follows, each type of the tested instances is described in the format $n_b - N - m_1 - m_2$, where n_b is the number of batches in the instance, N - the total number of jobs, m_1 and m_2 are the numbers of parallel machines on the first and the second stage, correspondingly. For each instance type, there were ten instances tested. The buffer capacity is denoted by $\Omega_h = hb_{max}$, where b_{max} is the maximum buffer requirement among all batches of an instance and h is an integer.

The first group of experiments aimed to compare the LR procedure, the permutation heuristic and CPLEX. It was also explored whether the scaling improves the performance of the procedure, and what scaling technique provides better results. In addition, it was investigated what order of batches, defined by starting times of batches obtained by Lagrangian relaxation, on the first or second stage, is more significant. The experiments were run for $5-50-5-2$, $5-100-5-2$ and $10-100-5-2$ instances for the buffer sizes Ω_2, Ω_3 and Ω_5. The permutations for the permutation heuristic were generated by Heap's algorithm [49]. The subgradient algorithm within the LR procedure was run for 150 iterations. The time limit for all tested algorithms

was 30 minutes.

The results presented in the Tables 3.17 - 3.25 compare the values of the objective function, provided by the LR procedure with and without scaling, by the permutation heuristic and CPLEX. The first column of each table is the test instance's number; columns $2-4$ and $5-7$ contain the results for the LR procedure, with WAIT algorithm using the order of batches defined by starting times of batches, provided by Lagrangian relaxation, on the first or second stage, correspondingly; columns with heading *no scaling*, $k = 0.0001$, *adjustable* contain results of the LR procedure with no scaling or with scaling coefficient of 0.0001 or the adjustable scaling coefficient, correspondingly; columns 8 and 9 contain the results provided by the permutation heuristic and CPLEX, correspondingly. The 10^{th} column signifies the status of CPLEX solution: either optimal, or the best integer solution found, or there is no integer solution found within the given time frame. Each value RV in columns $1-9$ is calculated as

$$RV = \frac{V - BestV}{BestV} \times 100\%,$$

where V signifies the smallest value of the objective function, provided by a feasible schedule constructed by the algorithm in the column title, and $BestV$ is the smallest value among the 8 algorithms. The results demonstrate that the LR procedure with scaling provided smaller values of the objective function (mostly within 5% of the best) than the procedure without scaling; the permutation heuristic provided the best results for larger instances of 100 jobs; CPLEX provided optimal or the best solutions for smaller instances with 50 jobs, however, did not provide any best solutions for instances with 100 jobs.

The box-plots depicted on the Figure 3-11 compare the tightness of solutions for the LR procedure with or without scaling for buffer sizes Ω_2, Ω_3 and Ω_5. For each instance a relative error was calculated as follows:

$$RE = \frac{UB - LB}{UB} \times 100\%,$$

where UB is the best value of the objective function provided by a feasible schedule,

Table 3.17: $5-50-5-2$, Ω_2 instances relative error from the best, in %

	LR-1st stage order			LR-2nd stage order					
Instance	No scaling	k=0.0001	Adjustable	No scaling	k=0.0001	Adjustable	Permut	CPLEX	CPLEX solution
0	19.96	8.58	8.58	82.81	9.46	**0.00**	**0.00**	239198.92	did not find a solution
1	19.58	**0.00**	0.15	19.58	**0.00**	1.38	1.38	**0.00**	solution exists
2	70.94	2.00	2.00	38.80	**0.00**	1.68	2.00	2.32	solution exists
3	29.29	2.36	2.62	18.41	2.36	2.62	2.62	**0.00**	solution exists
4	225.14	5.31	8.61	225.14	5.31	8.61	8.86	**0.00**	optimal
5	63.70	**0.00**	**0.00**	20.16	**0.00**	**0.00**	**0.00**	**0.00**	optimal
6	316.40	4.25	4.25	310.39	14.00	17.00	4.25	**0.00**	optimal
7	0.06	0.06	**0.00**	**0.00**	**0.00**	**0.00**	3.86	**0.00**	optimal
8	118.27	**0.00**	**0.00**	45.02	**0.00**	**0.00**	2.17	**0.00**	optimal
9	29.17	5.26	25.41	22.71	19.28	17.28	4.04	**0.00**	solution exists

Table 3.18: $5-50-5-2$, Ω_3 instances relative error from the best, in %

	LR-1st stage order			LR-2nd stage order					
Instance	No scaling	k=0.0001	Adjustable	No scaling	k=0.0001	Adjustable	Permut	CPLEX	CPLEX solution
0	117.21	0.50	0.50	126.65	0.50	0.50	0.50	**0.00**	optimal
1	21.06	1.24	1.39	21.06	1.24	1.39	2.63	**0.00**	optimal
2	11.40	2.07	2.07	43.39	2.07	2.07	2.07	**0.00**	optimal
3	44.66	0.26	0.26	44.66	**0.00**	0.26	0.26	2.30	solution exists
4	225.14	4.31	4.56	225.14	4.56	4.56	8.61	**0.00**	optimal
5	273.96	**0.00**	**0.00**	59.72	**0.00**	**0.00**	**0.00**	**0.00**	optimal
6	313.40	4.25	1.25	296.95	4.25	3.00	4.25	**0.00**	optimal
7	53.96	0.06	3.80	38.57	**0.00**	0.06	3.86	**0.00**	solution exists
8	115.54	**0.00**	**0.00**	49.20	**0.00**	**0.00**	2.17	**0.00**	optimal
9	29.17	**0.00**	2.00	3.21	2.00	2.00	3.21	**0.00**	optimal

Table 3.19: $5-50-5-2$, Ω_5 instances relative error from the best, in %

	LR-1st stage order			LR-2nd stage order					
Instance	No scaling	k=0.0001	Adjustable	No scaling	k=0.0001	Adjustable	Permut	CPLEX	CPLEX solution
0	116.13	**0.00**	**0.00**	125.52	**0.00**	**0.00**	**0.00**	10.46	solution exists
1	21.06	1.24	1.39	21.06	1.24	1.39	2.63	**0.00**	optimal
2	126.11	**0.00**	**0.00**	40.48	**0.00**	**0.00**	**0.00**	1.21	solution exists
3	44.66	0.26	0.26	44.66	0.26	0.26	0.26	**0.00**	solution exists
4	225.14	0.50	4.56	225.14	4.31	4.56	8.61	**0.00**	optimal
5	273.96	**0.00**	**0.00**	89.19	**0.00**	**0.00**	**0.00**	**0.00**	optimal
6	312.89	3.00	3.00	310.39	3.00	3.00	4.25	**0.00**	optimal
7	53.90	3.80	3.80	38.57	**0.00**	0.06	3.86	**0.00**	solution exists
8	115.54	**0.00**	**0.00**	49.20	**0.00**	**0.00**	2.17	**0.00**	optimal
9	29.17	**0.00**	2.00	3.21	2.00	2.00	3.21	**0.00**	optimal

Table 3.20: $5-100-5-2$, Ω_2 instances relative error from the best, in %

	LR-1st stage order			LR-2nd stage order					
Instance	No scaling	k=0.0001	Adjustable	No scaling	k=0.0001	Adjustable	Permut	CPLEX	CPLEX solution
0	3.66	**0.00**	**0.00**	3.66	**0.00**	**0.00**	**0.00**	30.03	solution exists
1	1.81	**0.00**	0.73	1.81	**0.00**	0.73	1.47	10224.84	did not find a solution
2	7.73	0.39	0.39	7.73	**0.00**	0.39	0.39	4749.51	did not find a solution
3	10.20	0.01	0.01	10.20	0.01	**0.00**	0.53	2.49	solution exists
4	12.36	0.26	**0.00**	12.36	0.26	**0.00**	0.26	19.72	solution exists
5	17.84	**0.00**	0.74	24.59	**0.00**	0.24	0.97	5561.49	did not find a solution
6	29.71	0.36	**0.00**	29.71	0.36	**0.00**	0.36	6372.69	did not find a solution
7	1.63	**0.00**	**0.00**	0.17	0.01	0.01	0.17	9451.17	did not find a solution
8	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	81.02	solution exists
9	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	34.37	solution exists

Table 3.21: $5-100-5-2$, Ω_3 instances relative error from the best, in %

	LR-1st stage order			LR-2nd stage order					
Instance	No scaling	k=0.0001	Adjustable	No scaling	k=0.0001	Adjustable	Permut	CPLEX	CPLEX solution
0	2.31	**0.00**	**0.00**	3.66	**0.00**	**0.00**	**0.00**	76.55	solution exists
1	38.14	0.90	**0.00**	38.14	0.91	0.73	1.47	3.44	solution exists
2	7.73	0.39	**0.00**	7.73	**0.00**	0.39	0.39	44.49	solution exists
3	10.26	0.52	0.53	9.87	0.13	**0.00**	0.53	34.99	solution exists
4	12.06	**0.00**	**0.00**	12.06	**0.00**	**0.00**	**0.00**	2.98	solution exists
5	2.99	0.97	0.74	16.86	0.48	**0.00**	0.97	36.33	solution exists
6	27.75	**0.00**	**0.00**	27.75	**0.00**	**0.00**	0.64	2.71	solution exists
7	10.15	1.22	1.21	8.73	0.02	1.21	1.37	**0.00**	solution exists
8	17.55	**0.00**	**0.00**	17.55	**0.00**	**0.00**	**0.00**	71.57	solution exists
9	14.97	**0.00**	**0.00**	14.97	**0.00**	**0.00**	**0.00**	72.94	solution exists

Table 3.22: $5-100-5-2$, Ω_5 instances relative error from the best, in %

	LR-1st stage order			LR-2nd stage order					
Instance	No scaling	k=**0.00**01	Adjustable	No scaling	k=**0.00**01	Adjustable	Permut	CPLEX	CPLEX solution
0	2.31	**0.00**	**0.00**	1.20	**0.00**	**0.00**	**0.00**	89.72	solution exists
1	36.91	**0.00**	**0.00**	36.91	**0.00**	**0.00**	0.56	6.54	solution exists
2	7.73	0.39	**0.00**	7.73	0.39	0.39	0.39	23.35	solution exists
3	10.53	**0.00**	0.53	10.39	0.53	**0.00**	0.53	54.91	solution exists
4	12.06	0.26	**0.00**	12.06	0.26	**0.00**	**0.00**	2.98	solution exists
5	38.32	0.74	**0.00**	38.32	0.74	0.74	0.97	35.22	solution exists
6	43.53	0.64	**0.00**	32.59	**0.00**	0.64	0.64	42.36	solution exists
7	10.85	0.46	0.46	6.31	**0.00**	0.46	0.46	3.33	solution exists
8	17.55	**0.00**	**0.00**	17.55	**0.00**	**0.00**	**0.00**	3.96	solution exists
9	14.97	**0.00**	**0.00**	14.97	**0.00**	**0.00**	**0.00**	72.94	solution exists

Table 3.23: $10-100-5-2$, Ω_2 instances relative error from the best, in %

	LR-1st stage order			LR-2nd stage order					
Instance	No scaling	k=**0.00**01	Adjustable	No scaling	k=**0.00**01	Adjustable	Permut	CPLEX	CPLEX solution
0	53.12	0.79	0.79	44.44	1.21	1.21	**0.00**	5954.00	did not find a solution
1	63.33	2.77	**0.00**	44.92	2.77	**0.00**	**0.00**	4760.30	did not find a solution
2	31.00	16.06	14.80	19.74	18.78	12.27	**0.00**	3487.81	did not find a solution
3	83.30	3.60	0.29	77.79	12.87	**0.00**	**0.00**	5037.96	did not find a solution
4	3.63	2.67	0.15	2.80	3.00	2.44	**0.00**	6055.28	did not find a solution
5	99.53	12.61	7.29	46.69	14.28	7.87	**0.00**	3025.54	did not find a solution
6	23.24	4.36	7.39	16.30	1.28	1.28	**0.00**	6568.36	did not find a solution
7	40.02	0.09	0.10	14.49	0.10	0.10	**0.00**	8915.47	did not find a solution
8	15.23	2.72	2.72	1.35	1.52	2.72	**0.00**	3604.25	did not find a solution
9	24.03	3.07	1.43	34.99	2.77	1.38	**0.00**	3914.56	did not find a solution

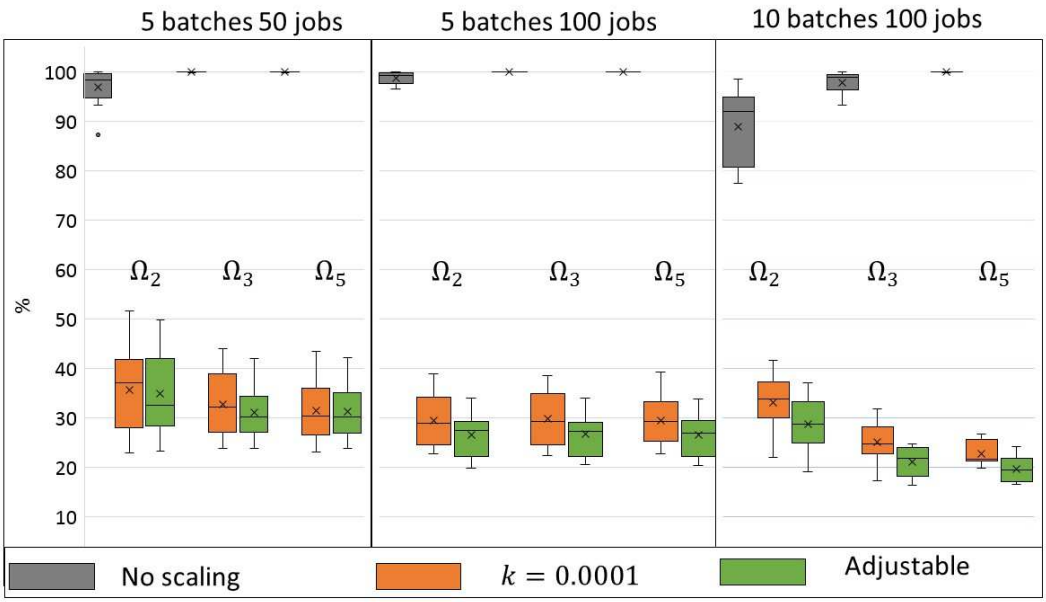
Table 3.24: $10-100-5-2$, Ω_3 instances relative error from the best, in %

	LR-1st stage order			LR-2nd stage order					
Instance	No scaling	k=**0.00**01	Adjustable	No scaling	k=**0.00**01	Adjustable	Permut	CPLEX	CPLEX solution
0	81.09	**0.00**	**0.00**	69.56	0.20	0.37	**0.00**	7.98	solution exists
1	123.08	**0.00**	**0.00**	75.14	**0.00**	**0.00**	**0.00**	5010.30	did not find a solution
2	33.60	1.91	3.67	25.24	5.21	4.10	**0.00**	3680.27	did not find a solution
3	169.59	5.38	1.24	106.04	3.80	2.27	**0.00**	5609.47	did not find a solution
4	65.32	1.07	2.29	7.17	**0.00**	0.02	0.02	6648.30	did not find a solution
5	58.43	0.59	0.06	21.54	1.92	1.92	**0.00**	153.57	solution exists
6	53.57	3.62	4.32	43.55	0.43	0.43	**0.00**	7418.98	did not find a solution
7	40.17	0.10	0.10	41.13	0.10	0.10	**0.00**	8923.81	did not find a solution
8	11.01	**0.00**	**0.00**	9.26	**0.00**	**0.00**	0.25	3832.09	did not find a solution
9	56.39	**0.00**	0.47	61.78	**0.00**	**0.00**	**0.00**	4706.26	did not find a solution

Table 3.25: $10-100-5-2$, Ω_5 instances relative error from the best, in %

	LR-1st stage order			LR-2nd stage order					
Instance	No scaling	k=**0.00**01	Adjustable	No scaling	k=**0.00**01	Adjustable	Permut	CPLEX	CPLEX solution
0	88.13	0.12	**0.00**	82.67	0.37	0.37	**0.00**	3.25	solution exists
1	72.24	**0.00**	**0.00**	54.79	**0.00**	**0.00**	**0.00**	26.75	solution exists
2	37.18	**0.00**	0.20	22.57	**0.00**	**0.00**	0.20	3777.17	did not find a solution
3	140.42	**0.00**	**0.00**	80.96	**0.00**	**0.00**	**0.00**	54.01	solution exists
4	108.54	0.29	0.38	8.94	0.38	0.38	**0.00**	6.11	solution exists
5	86.12	**0.00**	**0.00**	23.23	**0.00**	**0.00**	**0.00**	65.66	solution exists
6	97.44	**0.00**	0.43	72.24	**0.00**	**0.00**	**0.00**	16.41	solution exists
7	73.11	0.10	0.10	53.29	0.10	0.10	**0.00**	0.32	solution exists
8	114.56	0.90	0.90	79.28	0.90	0.90	1.15	**0.00**	solution exists
9	220.74	0.10	0.10	156.08	0.59	0.38	**0.00**	5.55	solution exists

Figure 3-11: Relative Error



constructed by the corresponding LR procedure with the order of jobs defined by the starting times of batches on the first stage, obtained by Lagrangian relaxation, and LB is the best value of (3.4.24). The smallest relative errors are provided by the LR procedure with the adjustable scaling coefficient provided the tightest solutions with less variability across all instances: for 50 job instances the relative errors vary between $23\% - 49\%$, for 100 job instances with 5 batches - between $20\% - 40\%$, for 100 job instances with 10 batches - between $20\% - 30\%$.

The Tables 3.26 - 3.28 compare the results provided by LR procedure with or without scaling when the order of batches is defined by the starting times of batches, obtained by Lagrangian relaxation, on the first or the second stage, correspondingly.

Table 3.26: Order on the 1st stage vs. order on the 2nd stage: $5-50-5-2$ instances, in %

	Ω_2			Ω_3			Ω_5		
Instance	No scaling	k=0.0001	Adjustable	No scaling	k=0.0001	Adjustable	No scaling	k=0.0001	Adjustable
0	-34.38	-0.81	8.58	-4.16	**0.00**	**0.00**	-4.16	**0.00**	**0.00**
1	**0.00**	**0.00**	-1.21	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
2	23.16	2.00	0.32	-22.31	**0.00**	**0.00**	60.95	**0.00**	**0.00**
3	9.19	**0.00**	**0.00**	**0.00**	0.26	**0.00**	**0.00**	**0.00**	**0.00**
4	**0.00**	**0.00**	**0.00**	**0.00**	-0.24	**0.00**	**0.00**	-3.65	**0.00**
5	36.23	**0.00**	**0.00**	134.13	**0.00**	**0.00**	97.66	**0.00**	**0.00**
6	1.46	-8.55	-10.90	4.14	**0.00**	**-1.70**	0.61	**0.00**	**0.00**
7	0.06	0.06	**0.00**	11.10	0.06	**3.74**	11.06	3.80	3.74
8	50.51	**0.00**	**0.00**	44.46	**0.00**	**0.00**	44.46	**0.00**	**0.00**
9	5.27	-11.75	6.93	25.15	-1.96	**0.00**	25.15	-1.96	**0.00**

Table 3.27: Order on the 1st stage vs. order on the 2nd stage: $5-100-5-2$ instances, in %

	Ω_2			Ω_3			Ω_5		
Instance	No scaling	k=0.0001	Adjustable	No scaling	k=0.0001	Adjustable	No scaling	k=0.0001	Adjustable
0	**0.00**	**0.00**	**0.00**	-1.30	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
1	**0.00**	**0.001**	**0.00**	**0.00**	-0.003	-0.72	**0.00**	**0.001**	**0.00**
2	**0.00**	0.39	**0.00**	**0.00**	0.39	-0.39	**0.00**	0.39	**0.00**
3	**0.00**	**0.00**	0.01	0.35	0.39	0.53	**0.00**	**0.00**	0.01
4	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
5	-5.42	**0.00**	0.51	-11.87	0.49	0.74	-5.42	**0.00**	0.51
6	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
7	1.46	-0.01	**-0.01**	1.30	1.20	**0.00**	1.46	-0.01	-0.01
8	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
9	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**

The experiments were run for $5-50-5-2$ and $10-100-5-2$ instances with buffer sizes Ω_2, Ω_3 and Ω_5. For each instance the comparative value CV, in %, was calculated as

$$CV = \left(\frac{1^{st}StageValue}{2^{nd}StageValue} - 1 \right) \times 100\%,$$

where $1^{st}StageValue$ and $2^{nd}StageValue$ are the best values of the objective function provided by the corresponding LR procedure with the order of batches defined by the starting times of batches, obtained by Lagrangian relaxation, on the first or the second stage, correspondingly. The results demonstrate, that for LR procedure with a scaling coefficient, the difference is not significant, with results with order the first or the second stage being within at most 3% from each other for the absolute majority of instances. However, the results for LR procedure without scaling show greater variability with differences up to 100%.

The Figures 3-12 and 3-13 represent the typical graphs of how the step τ and

Table 3.28: Order on the 1st stage vs. order on the 2nd stage: $10-100-5-2$ instances, in %

	Ω_2			Ω_3			Ω_5		
Instance	No scaling	k=0.0001	Adjustable	No scaling	k=0.0001	Adjustable	No scaling	k=0.0001	Adjustable
0	6.01	-0.41	-0.41	6.80	-0.20	-0.36	2.99	-0.24	-0.36
1	12.70	**0.00**	**0.00**	27.37	**0.00**	**0.00**	11.27	**0.00**	**0.00**
2	9.40	-2.29	2.25	6.67	-3.14	-0.41	11.92	**0.00**	0.20
3	3.10	-8.22	0.29	30.85	1.52	-1.01	32.86	**0.00**	**0.00**
4	0.81	-0.32	-2.24	54.26	1.07	2.27	91.43	-0.09	**0.00**
5	36.02	-1.46	-0.53	30.35	-1.31	-1.82	51.03	**0.00**	**0.00**
6	5.97	3.04	6.04	6.98	3.18	3.88	14.63	**0.00**	0.43
7	22.29	-0.01	**0.00**	-0.68	**0.00**	**0.00**	12.93	**0.00**	**0.00**
8	13.70	1.18	**0.00**	1.60	**0.00**	**0.00**	19.68	**0.00**	**0.00**
9	-8.12	0.30	0.05	-3.34	**0.00**	0.47	25.25	-0.49	-0.28

the relative error change with each iteration of the LR procedure. The provided graphs show the changes for $5-100-5-2$ instance with Ω_3 buffer, and the permutation defined by starting times of batches on the first stage, provided by Lagrangian relaxation, being used as the batches' order on both stages. On the first graph *no scaling*, $k = 0.0001$, *adjastable* signify values of the step τ for the LR procedure without no scaling, with the scaling coefficient $k = 0.0001$ and the adjustable scaling coefficient, correspondingly. On the second graph the *no scaling_LB*, 0.0001*_LB*, *adjustable_LB* signify the current values of lower bound (3.4.24) and *no scaling_UB*, 0.0001*_UB*, *adjustable_UB* signify the current values of upper bound (the value of the current feasible solution) provided at each iteration by the LR procedure without scaling, with the scaling coefficient $k = 0.0001$ and the adjustable scaling coefficient, correspondingly. The graphs explicitly demonstrate that the LR procedure with scaling provides more gradual decrease of the step τ than the LR procedure without scaling, and that the corresponding relative error and the upper bound - and hence the resulting value of the objective function - are significantly smaller.

The next group of experiments aimed to investigate the impact of the smaller horizon T: the LR procedure with and without scaling was run for the instances $5-50-5-2$ and $10-100-5-2$ with the buffer size Ω_2 using the larger planning horizon (3.4.1) and the smaller planning horizon (3.4.3); Tables 3.29 - 3.30 represent the results of these experiments. For each instance the relative difference in value and

Figure 3-12: Step τ: comparison of scaling and no scaling options

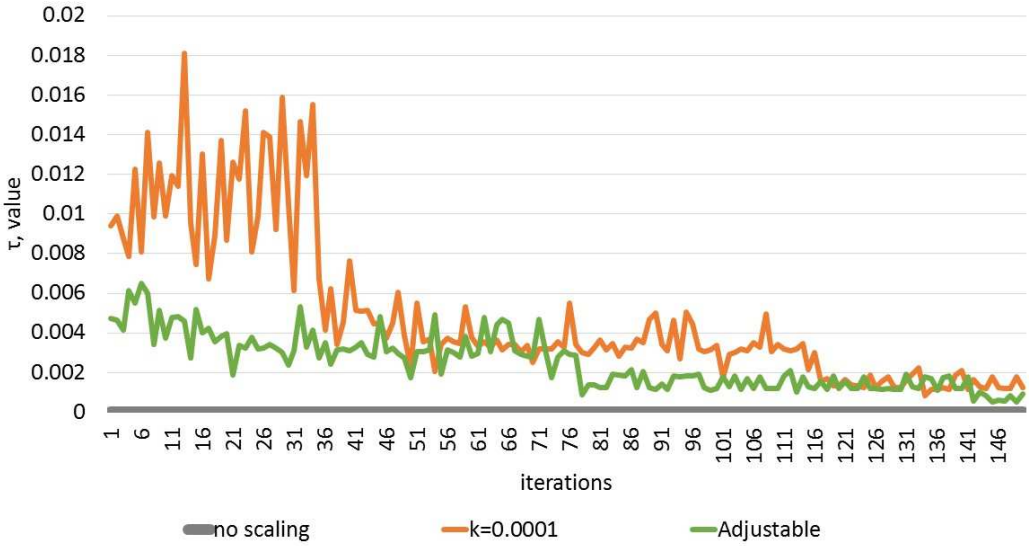


Figure 3-13: Convergence of the algorithms: upper and lower bounds

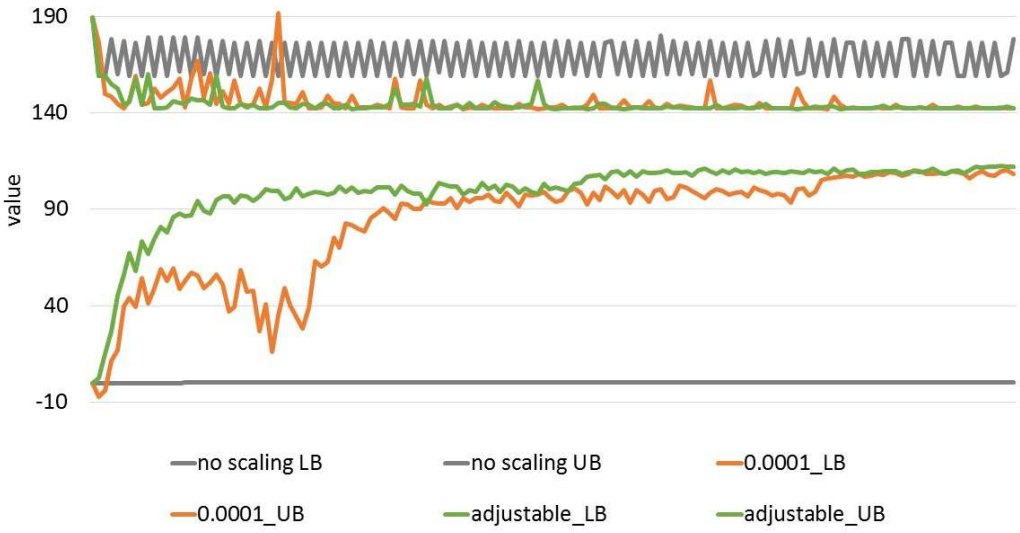


Table 3.29: Smaller planning horizon T: objective value and time, $5-50-2$, Ω_2

	Improvement, 1-NewValue/OldValue, in %			Improvement, 1-NewTime/OldTime, in %		
instance	no scaling	k=0.0001	Adjustable	no scaling	k=0.0001	Adjustable
0	38.2	-0.3	0.0	84.4	84.4	84.4
1	0.0	0.2	0.0	86.1	86.1	86.1
2	28.4	20.4	-1.7	84.0	84.0	84.0
3	0.0	0.3	0.0	84.8	84.8	84.8
4	0.0	3.0	-3.6	85.2	85.2	85.2
5	-31.2	0.0	0.0	84.9	84.9	84.9
6	-1.5	0.0	0.0	85.5	85.5	85.5
7	0.0	0.0	0.1	84.7	84.7	84.7
8	0.0	0.0	0.0	85.6	85.6	85.6
9	0.0	15.2	2.0	88.7	88.7	88.7

time were calculated as follows:

$$\left(1-\frac{NewValue}{OldValue}\right)\times 100\% \text{ and } \left(1-\frac{NewTime}{OldTime}\right)\times 100\%,$$

where $NewValue$ and $NewTime$ are the value of the objective function and the time spent to obtain the value with the smaller planning horizon, and $OldValue$ and $OldTime$ are the value of the objective function and the time spent to obtain the value with the larger planning horizon. In each table, column 1 signifies the instance's number, columns 2-4 represent an improvement in the value of the objective function, in %, provided by the LR procedure without scaling, with scaling coefficient $k = 0.0001$ and the adjustable scaling coefficient, correspondingly; columns 5-7 represent an improvement in time, in %, provided by the LR procedure without scaling, with scaling coefficient $k = 0.0001$ and the adjustable scaling coefficient, correspondingly. The change of the value of the objective function for LR procedure without scaling showed mixed results - for some instances the procedure with a smaller T provided a smaller value than the procedure with a larger T, for other instances - another way around. For the LR procedure with scaling the relative difference in the value of the objective function was not significant - within $0-4\%$ for most of instances. However, for all instances the time required to obtain a value of the objective function has improved dramatically with the smaller T - by $81\%-89\%$.

Another group of the experiments aimed to compare the LR procedure and per-

Table 3.30: Smaller planning horizon T: objective value and time, $10-100-5-2$, Ω_2

	Improvement, 1-NewValue/OldValue, in %			Improvement, 1-NewTime/OldTime, in %		
instance	no scaling	k=0.0001	Adjustable	no scaling	k=0.0001	Adjustable
0	-1.9	0.0	0.0	82.2	81.4	81.2
1	1.3	-1.4	0.0	82.5	81.6	79.1
2	6.9	-1.0	-4.5	81.5	81.6	79.8
3	18.9	0.2	-0.3	81.6	82.3	80.3
4	33.4	0.3	2.4	82.6	83.3	80.0
5	-1.7	0.1	4.2	82.2	81.5	80.1
6	18.0	0.0	0.0	81.5	81.3	81.0
7	-16.1	0.0	0.0	81.8	80.9	79.8
8	1.3	0.0	0.0	83.2	83.0	82.6
9	9.8	0.6	0.0	83.2	81.5	81.6

mutation heuristic for larger instances. The experiments were run for instances $10-200-5-2$, $10-300-5-2$, $10-500-5-2$ and $12-200-15-4$, $12-300-15-4$, $12-500-15-4$ and the buffer size Ω_5. The LR procedure utilised was the LR procedure with adjustable scaling coefficient and the order of batches, defined by the starting times of batches on the first stage and obtained by Lagrangian relaxation. The results are summarised in Tables 3.31 - 3.32. For the instances with 12 batches, the permutation heuristic was run for the first 44 million permutations, as the total number of the permutations is very large ($12! = 479,001,600$). The time limit for both algorithms was 30 min. For each instance the relative errors from the best value RE_1 and RE_2 were calculated as follows:

$$RE_1 = \frac{LR - Best}{Best} \times 100\% \text{ and } RE_2 = \frac{Permut - Best}{Best} \times 100\%,$$

where LR and $Permut$ are the values of the objective function provided by the LR procedure and permutation heuristic, correspondingly, and $Best$ is the best (smallest) value of the two. For the instances with 10 batches both heuristics provided values within 1% from each other. However, for the instances with 12 batches, the LR procedure provided the best values for all instances. In addition, for the instances with 300 jobs the permutation heuristic went through $36-37$ mln permutations within the given time, and for instances with 500 jobs - only through about 24 mln permutations.

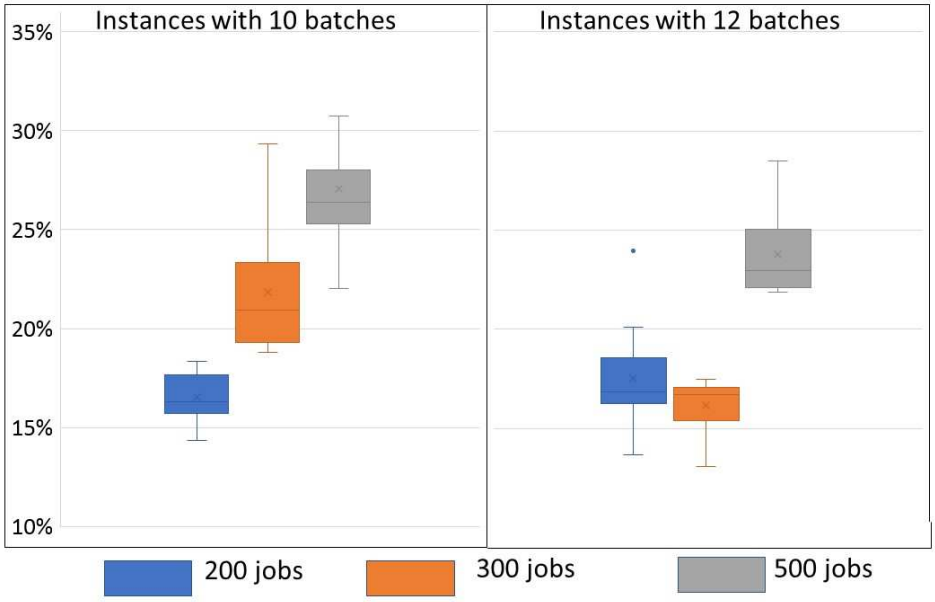
Table 3.31: Lagrangian heuristic vs. permutation heuristic: instances with 10 batches

	200 jobs		300 jobs		500 jobs	
instances	$\frac{LR-Best}{Best}$	$\frac{Permut-Best}{Best}$	$\frac{LR-Best}{Best}$	$\frac{Permut-Best}{Best}$	$\frac{LR-Best}{Best}$	$\frac{Permut-Best}{Best}$
0	**0.00**	0.14	**0.00**	0.01	**0.00**	**0.00**
1	**0.00**	**0.00**	**0.00**	0.11	**0.00**	**0.00**
2	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
3	0.39	**0.00**	0.00	**0.00**	**0.00**	**0.00**
4	**0.00**	0.01	0.00	**0.00**	**0.00**	0.04
5	**0.00**	0.11	**0.00**	0.21	**0.00**	0.10
6	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.18
7	**0.00**	0.04	0.17	**0.00**	**0.00**	**0.00**
8	0.26	**0.00**	0.20	**0.00**	**0.00**	**0.00**
9	**0.00**	**0.00**	0.17	**0.00**	**0.00**	0.16

Table 3.32: Lagrangian heuristic vs. permutation heuristic: instances with 12 batches

	200 jobs		300 jobs		500 jobs	
instances	$\frac{LR-Best}{Best}$	$\frac{Permut-Best}{Best}$	$\frac{LR-Best}{Best}$	$\frac{Permut-Best}{Best}$	$\frac{LR-Best}{Best}$	$\frac{Permut-Best}{Best}$
0	**0.00**	49.07	**0.00**	42.00	**0.00**	79.90
1	**0.00**	35.82	**0.00**	26.90	**0.00**	42.83
2	**0.00**	19.10	**0.00**	12.49	**0.00**	2.41
3	**0.00**	4.77	**0.00**	61.73	**0.00**	0.05
4	**0.00**	7.34	**0.00**	72.60	**0.00**	19.40
5	**0.00**	56.52	**0.00**	14.69	**0.00**	33.28
6	**0.00**	4.90	**0.00**	0.36	**0.00**	40.33
7	**0.00**	56.74	**0.00**	115.36	**0.00**	8.25
8	**0.00**	73.21	**0.00**	41.30	**0.00**	31.75
9	**0.00**	2.34	**0.00**	57.21	**0.00**	98.15

Figure 3-14: Relative Error - larger instances, Ω_5 buffer size



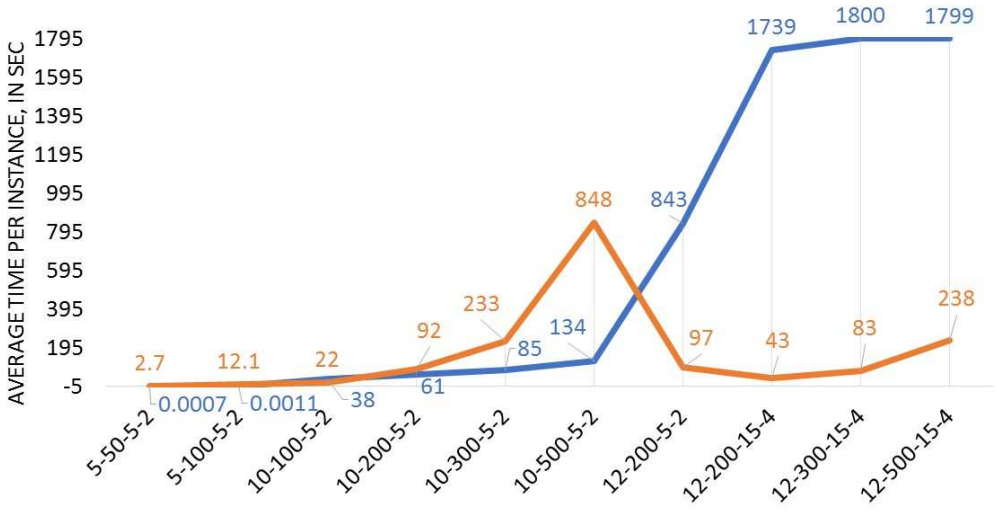
The box-plots depicted in Figure 3-14 show the relative errors, in %, provided by the LR procedure for the larger instances calculated as

$$RE = \frac{LR - LB}{LR} \times 100\%,$$

where LR is the value of the objective function provided by the LR procedure and LB is the best lower bound (largest), provided by Lagrangian relaxation. For larger instances with $200 - 500$ jobs the LR procedure provides even tighter solutions then for smaller instances with 50-100 jobs, with the relative error within $13 - 23\%$ for 200 instances; $13 - 29\%$ - for 300 jobs instances and $22 - 35\%$ - for the 500 jobs instances.

The graph in Figure 3-15 reflects the average time required to obtain the best feasible schedule for instances with various parameters provided by the LR procedure (with adjustable scaling coefficient and the order of batches defined by the starting times of batches on the first stage, obtained by Lagrangian relaxation) and permutation heuristic. The buffer size for all instances is Ω_5. The graph demonstrates that for LR procedure the time is affected by several factors and increases with the number

Figure 3-15: Average time per instance



of jobs, number of batches and may decrease if the number of machines on each stage increases. However, while the permutation heuristic is much faster for the instances with 50-100 jobs and up to 10 batches, the time increases dramatically, once the number of batches is greater than 10. In comparison, the CPLEX average time for $5-50-5-2$ instances was 24 min, and for $5-100-5-2$ and $10-100-5-2$ CPLEX utilised all available time (30 min) for each instance. Further, for $5-100-5-2$ instance with Ω_2 buffer size, in 5 hours CPLEX has improved the value obtained in 30 min by 14% only, however, the improved value was still 11.6% greater than the best value obtained by the LR procedure.

In summary, the LR optimisation procedure with a scaling coefficient performs efficiently for larger instances and generates better solutions, than the procedure without scaling; the scaling option allows to evaluate the quality of the solutions better, as the scaling allows to significantly improve the relative error. The strength of the improved LR optimisation procedure is demonstrated by the fact that the permutation approach or a straightforward integer programming approach failed to obtain good solutions for larger instances.

3.4.9 Conclusion

This section is concerned with a Lagrangian relaxation decomposition-based optimisation procedure (LR procedure) applied to the problem of minimisation of the total weighted tardiness for the two-stage flow shop with a job-dependent buffer. In this problem, each stage is represented by a number of parallel identical machines. The jobs are partitioned in predefined batches. Jobs's first operations are processed by one of parallel machines of the first stage, and jobs's second operations are processed in the predefined batches on a second-stage machine. The buffer requirement varies from batch to batch; the batch occupies the buffer continuously from the start of its its earliest job on a first-stage machine till the completion of the batch on the second-stage machine. The LR procedure is presented with two additional scaling options, and these options are compared computationally with each other and with a permutation heuristic. The computational experiments demonstrated that LR procedure with scaling provides tighter results in terms of the relative error, furthermore, for larger instances the LR procedure outperforms both the permutation heuristic and CPLEX in terms of the value of the objective function the resulting solutions and the time spent. A new smaller planning horizon is proposed which significantly improves the processing time.

Chapter 4

Discrete optimisation with polynomially detectable boundaries and restricted level sets

The results discussed in this chapter have been published in the following conference proceedings and a journal:

- [109]: Yakov Zinder, Julia Memar and Gaurav Singh. "Discrete optimisation with polynomially detectable boundaries and restricted level sets". Journal of Combinatorial Optimisation. 25-2, pp 308–325, 2013.
- [80]: Julia Memar, Gaurav Singh and Yakov Zinder. "Scheduling Partially Ordered UET Tasks on Dedicated Machines". IFAC Proceedings Volumes, 46(9), pp.1672-1677, 2013.
- [40]: Hanyu Gu, Julia Memar, Yakov Zinder. "Search Strategies for Problems with Detectable Boundaries and Restricted Level Sets". In Data and Decision Sciences in Action, pp. 149–162. Springer, 2018.

The results of this chapter have been presented at the following conferences:

- the 4th international conference on Combinatorial optimisation and applications COCOA2010, Hawaii, USA, 18-20 December 2010;

- the IFAC Conference on Manufacturing, Management and Control MIM 2013, St.Petersburg, Russian Federation, 19-21 June 2013;
- the 24th National Conference of the Australian Society for Operations Research ASOR2016, Canberra, Australia, 16-17 November 2016.

The computational experiments for this chapter were conducted on a personal computer with Intel Core $i5$ processor $CPU@1.70Ghz$, using Ubuntu 14.04 LTS, with base memory 4096 MB. The algorithms were implemented in C programming language.

4.1 Introduction

Consider the following discrete optimisation problem

$$\min_{(x_1,x_2,...,x_n)\in X} F(x_1, x_2, ..., x_n), \tag{4.1.1}$$

where $F(x_1, x_2, ..., x_n)$ is a function defined on an n-dimensional hypercube of points with integer coordinates satisfying the inequalities $0 \leq x_i \leq p(n)$, $1 \leq i \leq n$, where $p(n)$ is a polynomial in n. Without loss of generality, it is assumed that $p(n)$ is integer. The feasible region X is a subset of this hypercube. It is also assumed that $F(x_1, x_2, ..., x_n)$ is a nondecreasing function - for any $(x_1, ...x_n)$ and $(y_1, ...y_n)$, such that $x_i \geq y_i$, $1 \leq i \leq n$, $F(x_1, x_2, ..., x_n) \leq F(y_1, y_2, ..., y_n)$.

This description is too general for any specific optimisation procedure. The following three additional properties are introduced in this thesis to narrow the considered class of discrete optimisation problems. However, these properties are not very restrictive - the resultant class, for example, contains various well-known NP-hard problems of scheduling theory. These properties allow to describe a new discrete optimisation procedure for the discrete optimisation problems (4.1.1) that posses these properties. The discrete optimisation procedure can be viewed as a generalisation of the exact method, described in [112, 113] for scheduling problems on parallel machines. In what follows, the expression "in polynomial time" has the standard meaning that, the number of operations is bounded above by a polynomial in n, and this polynomial remains the same for all instances of (4.1.1). Similarly, the expression "cardinality is bounded above by some polynomial in n" implies, that for all instances of (4.1.1), the number of elements in the considered set is bounded above by the value of some polynomial in n and this polynomial remains the same for all instances of (4.1.1).

- The first property is concerned with the boundary of X the definition of which is based on the notion of dominance:

  - Point $a = (a_1, a_2, ..., a_n)$ dominates point $b = (b_1, b_2, ..., b_n)$ if $b_i \leq a_i$ for all $1 \leq i \leq n$, and a strictly dominates b if at least one of these inequalities is

strict.

- The boundary of X is the set of all points in X which do not strictly dominate any point in X.

Property 1 There is an algorithm which for any point in X in polynomial time determines whether or not this point is on the boundary of X.

- The second property pertains to the notion of a level set defined as follows:

  - For any value $\bar{F}$ of F, a set D in the domain of F is $\bar{F}$-dominant if $F(x_1, x_2, ..., x_n) = \bar{F}$ for all $(x_1, x_2, ..., x_n) \in D$ and $F(y_1, y_2, ..., y_n) = \bar{F}$ implies that there exists a point in D which dominates $(y_1, y_2, ..., y_n)$.
  - For any value $\bar{F}$ of F, a level set, denoted by $A(\bar{F}, F)$, is an $\bar{F}$-dominant set with the smallest cardinality among all $\bar{F}$-dominant sets. The next section justifies this definition by showing that for any value $\bar{F}$ of F the corresponding level set is unique.

Property 2 For any value $\bar{F}$ of F, the corresponding level set can be found in polynomial time.

This property implies that the cardinality of each level set is bounded above by some polynomial in n and this polynomial remains the same for all instances of (4.1.1) - because otherwise the *Property 2* would not be satisfied. Observe that if $\bar{F} = F(p(n), ..., p(n))$, then $A(\bar{F}, F)$ is comprised of only one point $(p(n), ..., p(n))$.

- The third property is the existence of a polynomial-time algorithm that for any value $\bar{F}$ of F such that $\bar{F} < F(p(n), ..., p(n))$ finds the smallest value of F greater than $\bar{F}$.

Property 3 For any value $\bar{F}$ of F such that $\bar{F} < F(p(n), ..., p(n))$ the value

$$F' = \min_{\{(x_1,x_2,...,x_n):F(x_1,x_2,...,x_n)>\bar{F}\}} F(x_1, x_2, ..., x_n).$$

can be found in polynomial time.

4.2 Level sets

The theorem below justifies the definition of a level set by establishing its uniqueness. This theorem is based on the following lemma.

Lemma 1 *For any value $\bar{F}$ of F and any $\bar{F}$-dominant set D, $A(\bar{F}, F) \subseteq D$.*

Proof: Consider an arbitrary $a \in A(\bar{F}, F)$. Since D is an $\bar{F}$-dominant set, there exists $x \in D$ which dominates a. Since $A(\bar{F}, F)$ is also a dominant set, there exists $a' \in A(\bar{F}, F)$ which dominates x. If x strictly dominates a or a' strictly dominates x, then a' strictly dominates a. In this case the set $A(\bar{F}, F) - \{a\}$ is a dominant set, because any point dominated by a is also dominated by a', which contradicts the definition of a level set. Therefore $a = x$, and thus $A(\bar{F}, F) \subseteq D$. □

Let D be any set of points in the considered hypercube (the domain of F). Denote by D^c the set of all points $x \in D$ such that there is no point in D that strictly dominates x.

Theorem 1 *For any value $\bar{F}$ of F the level set is unique, and for any $\bar{F}$-dominant set D, $D^c = A(\bar{F}, F)$.*

Proof: Suppose that for some value $\bar{F}$ of F there exist two different level sets A and B. Then by Lemma 1, $A \subseteq B$ and $B \subseteq A$ which contradicts the assumption that A and B are different.

Since D is an $\bar{F}$-dominant set, for any $x = (x_1, ..., x_n)$ such that $F(x_1, ..., x_n) = \bar{F}$, there exists $d \in D$ which dominates x. If $d \notin D^c$, then there exists $d' \in D^c$ such that d' strictly dominates d, and therefore d' strictly dominates x. Hence, D^c is an

$\bar{F}$-dominant set and by Lemma 1, $A(\bar{F}, F) \subseteq D^c$. Suppose that there exists $d \in D^c$ such that $d \notin A(\bar{F}, F)$. Since $A(\bar{F}, F)$ is an $\bar{F}$-dominant set, there exists $a \in A(\bar{F}, F)$ such that a strictly dominates d, which by virtue of $A(\bar{F}, F) \subseteq D^c$ contradicts the definition of D^c. $\square$

4.3 Examples of the problems with the considered properties

4.3.1 *Property 1*: scheduling on dedicated machines - the boundary of the feasible region

The properties introduced in Section 4.1 are possessed by many NP-hard scheduling problems. For example, consider the following scheduling problem. A set of n tasks $N = \{1, \ldots, n\}$ is processed on a set $M = \{1, ..., m\}$ of m parallel machines subject to precedence constraints in the form of an anti-reflexive, anti-symmetric and transitive relation on N. If in this relation task i precedes task j, denoted $i \rightarrow j$, then task i must be completed before task j can be processed. If $i \rightarrow j$, then i is called a predecessor of j and j is called a successor of i. The processing time of each task is one unit of time. For each $j \in N$, the processing of task j can commence not earlier than its release time r_j, where r_j is a nonnegative integer. Without loss of generality it is assumed that the smallest release time is zero.

A machine can process only one task at a time and a task can be processed by only one machine at a time. Each task $j \in N$ can be processed only by a machine from a particular subset $M_j \subseteq M$, which will be referred to as a set of dedicated machines corresponding to task j. In what follows, it is assumed that the sets of dedicated machines are embedded, that is for any $j \in N$ and $g \in N$ either $M_j \subseteq M_g$, or $M_g \subseteq M_j$, or $M_j \cap M_g = \emptyset$. If a machine starts processing a task, it continues until completion, i.e. no preemptions are allowed.

The availability of machines is a function of time. More specifically, for any positive integer t, let $M(t)$ be the set of machines that can process tasks during the

time interval $[t-1,t]$. It is assumed that, for any $j \in N$ and any positive integer t, $M_j \cap M(t) \neq \emptyset$.

Since preemptions are not allowed, a schedule is specified by the tasks' completion times which without loss of generality are assumed being integers. In the scheduling literature the completion time of task j is usually denoted by C_j, but for the purpose of our discussion it is convenient to denote the completion time of task j by x_j. The goal is to minimize $F(x_1,...,x_n)$, where F is a nondecreasing function.

In the three field notation (see, for example, [87]) the above scheduling problem can be denoted by $P|prec, r_j, p_j = 1, M_j, emb, M(t)|F$, where P and $M(t)$ indicate that tasks are processed on parallel machines and that the set of available machines is a function of time, $prec$ and r_j specify that tasks are subject to precedence constraints and release times, $p_j = 1$ shows that all processing times are one unit of time, and M_j and emb state that each task can be processed only by a machine from the corresponding set of dedicated machines and the sets of dedicated machines are embedded. In the particular case, when each machine is available at any point in time, the parameter $M(t)$ is omitted. The particular case, when all release times are equal to zero, is specified by omitting r_j. Similarly, if the parameter M_j is omitted, then it is assumed that all machines are identical. If the precedence constraints are restricted to a certain type of graphs, then parameter $prec$ is replaced by a reference to this type of graphs. An alternative notation, as in the case of the job shop (see below), also can be used.

The $P|prec, r_j, p_j = 1, M_j, emb, M(t)|F$ problem includes as particular cases such well known scheduling problems [87] as

- $P|prec, r_j, p_j = 1, m(t)|F$ - the scheduling problem with parallel identical machines with the number $m(t)$ being a function of time, i.e. $m(t)$ is the number of identical machines available in the time interval $[t-1,t]$;

- $J|p_j = 1|F$ - the job shop scheduling problem with unit execution time tasks: $|M_j| = 1$ for any $j \in N$ and the precedence constraints are a collection of disjoint chains.

Even very restricted versions of these particular cases remain $NP-$hard in the strong sense. For example, $P|bipartite, p_j = 1|C_{max}$, where the precedence constraints are in the form of a bipartite graph and

$$F(x_1, ..., x_n) = \max_{1 \leq j \leq n} x_j$$

(the so called makespan objective function denoted by C_{max}), is $NP-$hard in the strong sense [110]. Another example is $J2|p_j = 1|C_{max}$, where $J2$ indicates that the tasks are to be scheduled on two machines. The NP-hardness of $J2|p_j = 1|C_{max}$ was established in [54].

Since $i \rightarrow j$ implies $x_i + 1 \leq x_j$, without loss of generality it is assumed that $i \rightarrow j$ implies $r_i + 1 \leq r_j$. In turn, the above assumption allows to assume that for any task i with $r_i > 0$, the number of tasks j with $r_j < r_i$ is greater than r_i. Indeed, suppose that this assumption does not hold, and among all i, violating this assumption, g is a task with the smallest release time. Then, even processing only one task at a time, one can complete all tasks j with $r_j < r_g$ before time r_g, and therefore the problem can be split into two separate problems: one with all tasks j satisfying $r_j < r_g$ and another with all remaining tasks. The above assumption implies that there exists an optimal schedule with all completion times less than or equal to n. Consequently, it is sufficient to consider the objective function F only on the n-dimensional hypercube of points $(x_1, ..., x_n)$ with integer coordinates satisfying $0 \leq x_j \leq n$ for all $1 \leq j \leq n$. Then, the feasible region X can be viewed as the set of points $(x_1, x_2, ..., x_n)$ corresponding to all feasible schedules with completion times less than or equal to n. In other words, X is the set of all points $(x_1, ..., x_n)$ with integer coordinates satisfying the following three conditions:

(a) $r_j + 1 \leq x_j \leq n$ for all $1 \leq j \leq n$;

(b) $|\{i : \ x_i = t \text{ and } M_i \subseteq M_j\}| \leq |M_j \cap M(t)|$ for all integer $1 \leq t \leq n$ and all j such that $x_j = t$;

(c) $x_i \leq x_j - 1$ for all i and j such that $i \rightarrow j$.

The following lemma shows that the $P|prec, r_j, p_j = 1, M_j, emb, M(t)|F$ scheduling problem has *Property 1*, as the condition specified in this lemma can be checked in $O(n^2)$ operations.

Lemma 2 *A point $(x_1, ..., x_n) \in X$ is on the boundary of X if and only if, for each integer $t \geq 1$ such that $|\{g : \ x_g = t\}| < |M(t)|$ and for each $x_j > t$, at least one of the following conditions holds:*

- *$r_j \geq t$;*
- *there is i such that $i \to j$ and $x_i \geq t$;*
- *there is M_i such that $M_j \subseteq M_i$ and $|\{g : \ x_g = t \text{ and } M_g \subseteq M_i\}| = |M_i \cap M(t)|$.*

Proof: Suppose that $x = (x_1, ..., x_n) \in X$ is on the boundary of X, and let $t \geq 1$ be any integer such that $|\{g : \ x_g = t\}| < |M(t)|$. Let j be any task such that $x_j > t$. Consider the point $x' = (x'_1, ..., x'_n)$, where $x'_j = t$ and $x'_g = x_g$ for all $g \neq j$. Since x is on the boundary of X and x strictly dominates x', $x' \notin X$. Hence, either $r_j \geq t$ or x' does not satisfy either feasibility condition (b) or (c) or both. If $r_j \geq t$, then the desired property holds. Suppose $r_j < t$ and x' does not satisfy the feasibility condition (c). This implies that there exists $i \to j$ and $x_i \geq t$. If $r_j < 0$ and there is no i such that $i \to j$ and $x_i \geq t$, then the feasibility condition (b) does not hold for x'. Thus, there exists M_i such that $M_j \subseteq M_i$ and $|\{g : \ x_g = t \text{ and } M_g \subseteq M_i\}| = |M_i \cap M(t)|$.

Conversely, consider $x = (x_1, ..., x_n) \in X$ and suppose that, for each integer $t \geq 1$ such that $|\{g : \ x_g = t\}| < |M(t)|$ and each j such that $x_j > t$, either $r_j \geq t$, or there exists i such that $x_i \geq t$ and $i \to j$, or there exists M_i such that $M_j \subseteq M_i$ and $|\{g : \ x_g = t \text{ and } M_g \subseteq M_i\}| = |M_i \cap M(t)|$. Suppose that x is not on the boundary of X, i.e. x strictly dominates some $x' = (x'_1, ..., x'_n) \in X$. Then, among all g such that $x'_g < x_g$ select one with the smallest x'_g. Let it be task j. Then, $x_g = x'_j$ implies $x'_g = x_g$. Hence $|\{g : \ x_g = x'_j\}| < |M(x'_j)|$, which together with $x'_j < x_j$ implies that one of the conditions of the lemma holds and therefore one of the feasibility conditions is violated. Thus, the inequality $r_j \geq x'_j$ contradicts $(x'_1, ..., x'_n) \in X$. The existence of i such that $x_i \geq x'_j$ and $i \to j$ also contradicts $x' \in X$ because $x'_i \geq x'_j$. Finally, if

for some i with $x_i = x'_j$ the corresponding M_i is such that $M_j \subseteq M_i$ and $|\{g :\ x_g = x'_j$ and $M_g \subseteq M_i\}| = |M_i \cap M(x'_j)|$, then x' does not satisfy the feasibility condition (b). □

4.3.2 *Property 2* and *Property 3*: level sets of $\max_{1\le j\le n} \varphi_j(x_j)$

Consider (4.1.1) with the objective function

$$F(x_1, x_2, ..., x_n) = \max_{1\le j\le n} \varphi_j(x_j), \tag{4.3.1}$$

where each $\varphi_j(x_j)$ is a nondecreasing function defined for all integer $0 \le x_j \le p(n)$.

- Let $\bar{F}$ be an arbitrary value of F, and let a_j be the largest among all integer x_j satisfying the inequalities $\varphi_j(x_j) \le \bar{F}$ and $x_j \le p(n)$. It is easy to see that $F(a_1, ..., a_n) = \bar{F}$. Moreover, if $F(x_1, x_2, ..., x_n) = \bar{F}$, then $(a_1, ..., a_n)$ dominates $(x_1, x_2, ..., x_n)$. Hence, for each $\bar{F}$ the level set $A(\bar{F}, F)$ is comprised of only one point. This point can be found in polynomial time, for example by using the binary search on the interval $[0, p(n)]$ separately for each φ_j. Hence, the objective function (4.3.1) has *Property 2*.

- Let $F' < F''$ be two consecutive values of F, i.e. there is no value of F between these two values. Let $(a'_1, ..., a'_n)$ and $(a''_1, ..., a''_n)$ be the points constituting $A(F', F)$ and $A(F'', F)$, respectively. Let J be the set of all j satisfying $a'_j < p(n)$. Observe that $(a''_1, ..., a''_n)$ strictly dominates $(a'_1, ..., a'_n)$ and $a'_j < a''_j$ implies $j \in J$. Moreover, for all $j \in J$, $\varphi_j(a'_j + 1) > F'$ and therefore $\varphi_j(a'_j + 1) \ge F''$. The above observations lead to the following inequalities

$$F'' \le \min_{j\in J} \varphi_j(a'_j + 1) \le \max_{1\le j\le n} \varphi_j(a''_j) = F''.$$

  Hence,

$$F'' = \min_{j\in J} \varphi_j(a'_j + 1). \tag{4.3.2}$$

  As has been shown above, for a given F', the corresponding point $(a'_1, ..., a'_n)$,

constituting $A(F', F)$, can be found in polynomial time. Then, F'' can be obtained using (4.3.2). Therefore, the objective function (4.3.1) has *Property 3*.

Objective functions of the form (4.3.1) are common in scheduling theory. Thus, one of the most frequently used objective functions in scheduling is (4.3.1) with all $\varphi_j(t) = t - d_j$, where d_j is interpreted as a due date of task j. In this case, the objective function is referred to as the maximum lateness denoted by L_{max}. If all due dates are zero, the maximum lateness problem converts into the makespan problem with the makespan objective function denoted by C_{max}.

Since the domain of each φ_j is the set of all integer points in the interval $[0, p(n)]$, φ_j takes on at most $p(n) + 1$ different values. Consequently, the cardinality of the range of (4.3.1) cannot exceed $n(p(n) + 1)$. So, the union of all level sets cannot contain more than $n(p(n) + 1)$ points. Starting with value $F(0, ..., 0)$ and with the corresponding level set (which is comprised of only one point), one can enumerate the union of all level sets in polynomial time. Nevertheless, in general, the problem remains NP-hard because the elements of level sets do not necessarily belong to the feasible region X. Thus, it is well known that the $P|prec, p_j = 1|C_{max}$ problem is NP-hard in the strong sense [71, 101]. Further, the problem remains NP-hard in the strong sense even if the precedence constraints are restricted to the bipartite graphs [110].

4.3.3 *Property 2* and *Property 3* in multi-objective optimisation

Consider an optimisation problem with k nondecreasing objective functions $F_1, ..., F_k$, each defined on the same n-dimensional hypercube of points with integer coordinates satisfying the inequalities $0 \leq x_i \leq p(n)$, $1 \leq i \leq n$, where $p(n)$ is a polynomial in n. A common approach in multi-objective optimisation is the replacement of several objective functions by a single function

$$F(x_1, ..., x_n) = \psi(F_1(x_1, ..., x_n), ..., F_k(x_1, ..., x_n)), \tag{4.3.3}$$

where ψ is a nondecreasing function.

Lemma 3 *For any value $\bar{F}$ of (4.3.3) and for any $(a_1, ..., a_n) \in A(\bar{F}, F)$, there are points $(a_1^{(i)}, ..., a_n^{(i)}) \in A(F_i(a_1, ..., a_n), F_i)$, $1 \leq i \leq k$, such that $a_j = \min\limits_{1 \leq i \leq k} a_j^{(i)}$ for all $1 \leq j \leq n$.*

Proof: Consider an arbitrary point $(a_1, ..., a_n) \in A(\bar{F}, F)$, and for each $1 \leq i \leq k$ denote $\bar{F}_i = F_i(a_1, ..., a_n)$. By the definition of $A(\bar{F}_i, F_i)$, there exists $(a_1^{(i)}, ..., a_n^{(i)}) \in A(\bar{F}_i, F_i)$ such that $a_j \leq a_j^{(i)}$ for all $1 \leq j \leq n$. Consider the point $(\tilde{a}_1, ..., \tilde{a}_n)$, where $\tilde{a}_j = \min\limits_{1 \leq i \leq k} a_j^{(i)}$ for all $1 \leq j \leq n$. Since ψ is a nondecreasing function, each F_i is a nondecreasing function, each $(a_1^{(i)}, ..., a_n^{(i)})$ dominates $(\tilde{a}_1, ..., \tilde{a}_n)$, and $(\tilde{a}_1, ..., \tilde{a}_n)$ dominates $(a_1, ..., a_n)$,

$$F(a_1, ..., a_n) \leq F(\tilde{a}_1, ..., \tilde{a}_n) \leq \psi\left(F_1(a_1^{(1)}, ..., a_n^{(1)}), ..., F_k(a_1^{(k)}, ..., a_n^{(k)})\right)$$
$$= \psi\left(\bar{F}_1, ..., \bar{F}_k\right) = F(a_1, ..., a_n).$$

Hence, $F(\tilde{a}_1, ..., \tilde{a}_n) = F(a_1, ..., a_n)$. On the other hand, by the definition of $A(\bar{F}, F)$, $F(a_1, ..., a_n) = \bar{F}$, and therefore $F(\tilde{a}_1, ..., \tilde{a}_n) = \bar{F}$. Moreover, by the same definition, there exists a point $(a_1', ..., a_n') \in A(\bar{F}, F)$ which dominates $(\tilde{a}_1, ..., \tilde{a}_n)$. Consequently, $a_j \leq \tilde{a}_j \leq a_j'$ for all $1 \leq j \leq n$. If at least one of these inequalities is strict, then $(a_1', ..., a_n')$ strictly dominates $(a_1, ..., a_n)$ which contradicts the definition of a level set because in this case $A(\bar{F}, F) - \{(a_1, ..., a_n)\}$ is an $\bar{F}$-dominant set. So, $a_j = \min\limits_{1 \leq i \leq k} a_j^{(i)}$ for all $1 \leq j \leq n$. □

According to Lemma 3 all level sets of F can be obtained from the level sets of F_1,..., F_k. Therefore, in some cases, the fact that each of F_1,..., F_k has *Property 2* or *Property 3* or both may imply that (4.3.3) also has these properties. Theorems 2 and 3 are concerned with such a case.

Theorem 2 *If each of F_1,..., F_k has Property 3 and the cardinality of the range of each F_1,..., F_k is bounded above by a polynomial in n, then (4.3.3) has Property 3.*

Proof: Each F_i is a nondecreasing function defined on the n-dimensional hypercube of points with integer coordinates satisfying the inequalities $0 \leq x_i \leq p(n)$, $1 \leq$

$i \leq n$. Therefore, its smallest value is $F_i(0, ..., 0)$. Since F_i has *Property 3* and the cardinality of the range of F_i is bounded above by some polynomial in n, it is possible to enumerate all values of F_i in polynomial time by starting with $F_i(0, ..., 0)$ and using *Property 3* of F_i. Consequently, it is possible in polynomial time to generate all combinations $(\bar{F}_1, ..., \bar{F}_k)$, where each $\bar{F}_i$ is some value of the corresponding F_i. These combinations give all values of (4.3.3), and therefore, (4.3.3) has *Property 3.* □

Theorem 3 *If each of F_1,..., F_k has Property 2 and Property 3 and the cardinality of the range of each F_1,..., F_k is bounded above by a polynomial in n, then (4.3.3) has Property 2.*

Proof: Let $\bar{F}$ be an arbitrary value of F, and let $(\bar{F}_1, ..., \bar{F}_k)$ be an arbitrary combination of values of F_1, ..., F_k such that $\bar{F} = \psi(\bar{F}_1, ..., \bar{F}_k)$. Since each F_i has *Property 2*, there exists an algorithm which in polynomial time finds all elements of $A(\bar{F}_i, F_i)$. Hence, it is possible to find in polynomial time all combinations $(a^{(1)}, ..., a^{(k)})$, where each $a^{(i)} = (a_1^{(i)}, ..., a_n^{(i)})$ is an element of the corresponding $A(\bar{F}_i, F_i)$. Each combination $(a^{(1)}, ..., a^{(k)})$ gives the point $(\min\limits_{1 \leq i \leq k} a_1^{(i)}, ..., \min\limits_{1 \leq i \leq k} a_n^{(i)})$. So, the cardinality of the set $D(\bar{F}_1, ..., \bar{F}_k)$ of all such points is bounded above by some polynomial in n.

Since the cardinality of the range of each F_i is bounded above by some polynomial in n and since each F_i has *Property 3*, it is possible to find in polynomial time all combinations $(\bar{F}_1, ..., \bar{F}_k)$ of values of F_1,..., F_k satisfying the condition $\bar{F} = \psi(\bar{F}_1, ..., \bar{F}_k)$. Furthermore, as has been shown above, there exists a polynomial-time algorithm which for each such combination finds all elements of the corresponding set $D(\bar{F}_1, ..., \bar{F}_k)$. Therefore, the union D of $D(\bar{F}_1, ..., \bar{F}_k)$ for all combinations $(\bar{F}_1, ..., \bar{F}_k)$, satisfying $\bar{F} = \psi(\bar{F}_1, ..., \bar{F}_k)$, can be found in polynomial time. According to Lemma 3, $A(\bar{F}, F) \subseteq D$, and therefore D is an $\bar{F}$-dominant set. Then, by Theorem 1, D^c is the level set. Since the cardinality of D is bounded above by a polynomial in n, D^c can be found in polynomial time which implies *Property 2.* □

4.4 Description of the Discrete Optimisation Procedure

4.4.1 Introduction

The discrete optimisation procedure is an iterative algorithm that at each iteration uses some lower bound on the optimal value of F. All lower bounds belong to the range of F. Of course, $F(0, ..., 0)$ can be taken as an initial lower bound, but a tighter bound may improve the convergence. At each iteration with some lower bound $\bar{F}$, the optimisation procedure searches for a feasible point dominated by one of the elements constituting $A(\bar{F}, F)$. *Property 2* guarantees that all elements of $A(\bar{F}, F)$ can be enumerated in polynomial time, and the optimisation procedure uses them in succession. If a feasible point, dominated by some element of $A(\bar{F}, F)$, has been found, the optimisation procedure terminates with this point as an optimal solution. Otherwise, using *Property 3*, the optimisation procedure finds a new lower bound and starts a new iteration.

Denote by F_0 the initial lower bound, and by $\bar{F}$ - the current lower bound. Let $NextLB(\bar{F})$ be a procedure that determines the next lower bound greater than the current lower bound. Assume that the procedure $LevelSet(\bar{F})$ determines the level set $A(\bar{F}, F)$ for the current lower bound, and the procedure $Dominate(a)$ determines whether or not the element $a \in A(\bar{F}, F)$ dominates any feasible point, and if it does, then the procedure returns $True$; if the procedure determines that a does not dominate any $x \in X$, it returns $False$. Let NI_{max} be the maximum number of iterations. The discrete optimisation procedure can be described as follows:

Discrete Optimisation Procedure

1: **Set** $k = 0$, $\bar{F} = F_0$, $Found = False$;
2: **while not** $Found$ **and** $k < NI_{max}$ **do**
3: $LevelSet(\bar{F})$, **Set** $i = 1$;
4: **while not** $Found$ **and** $i \leq |A(\bar{F}, F)|$ **do**
5: $a = A(\bar{F}, F)[i]$;

6: $Found = Dominate(a)$;

7: $i + 1$;

8: **end while**

9: **if not** $Found$ **then**

10: **Set** $\bar{F} = NextLB(\bar{F})$;

11: $k = k + 1$;

12: **end if**

13: **end while**

Search for a feasible point, dominated by an element of $A(\bar{F}, F)$, can be conducted in several different ways. This gives a rise to the three different approaches - descending, ascending and descending-ascending search methods. In all three approaches, each $(a_1, ..., a_n) \in A(\bar{F}, F)$ initiates a search tree, where all nodes correspond to points in the domain of F. The difference between the descending, ascending and descending-ascending approaches is in the method of constructing the search tree. These three methods are presented below.

4.4.2 Descending method

As it has been mentioned above, the root of the search tree corresponds to some $(a_1, ..., a_n) \in A(\bar{F}, F)$. At each stage of construction of the search tree, the optimisation procedure chooses a node that does not have successors in the already constructed fragment of this tree and connects this node to one or several new nodes (branching). The new nodes correspond to the points *dominated* by the point corresponding to the node at which branching occurs. Let $d = (d_1, ..., d_n) \notin X$ be a point which corresponds to a node without successors in the partially constructed search tree. Let

$$\varrho(d_1, ..., d_n) = \min_{(x_1,...,x_n)\in X} \max_{1\le j\le n} [x_j - d_j].$$

Point d dominates at least one feasible point if and only if

$$\varrho(d_1, ..., d_n) \le 0. \tag{4.4.1}$$

In general, the question whether or not $\varrho(b_1, ..., b_n) \leq 0$ is an NP-complete problem. Thus, the NP-completeness in the strong sense of this question for the $P|prec, p_j = 1|C_{max}$ scheduling problem, which is a particular case of (4.1.1), follows from [71] and [101]. Hence, instead of $\varrho(d_1, ..., d_n)$, one may attempt to calculate some $\underline{\varrho}$ such that

$$\underline{\varrho} \leq \varrho(d_1, ..., d_n). \tag{4.4.2}$$

The method of calculating $\underline{\varrho}$ depends on F and X. If $\underline{\varrho} > 0$, then d is fathomed, i.e.no branching at d is required. If $\underline{\varrho} \leq 0$ or $\underline{\varrho}$ has not been calculated at all, then d can be projected onto X, where the projection of d onto X is a point with the smallest t among all points $(d_1 + t, ..., d_n + t)$ satisfying $(d_1 + t, ..., d_n + t) \in X$. The point $(d_1, ..., d_n)$ can be projected onto X in polynomial time, since this requires that only the points $(d_1 + t, ..., d_n + t)$ with integer t satisfying the inequality $|t| \leq p(n)$ are considered. Of course, the projection may not exist. Consider the following cases:

- The projection $(d_1 + \tau, ..., d_n + \tau)$ exists and $\tau \leq 0$;
- The projection $(d_1 + \tau, ..., d_n + \tau)$ is on the boundary of X and $\tau > 0$;
- The projection $(d_1 + \tau, ..., d_n + \tau)$ is not the boundary of X and $\tau > 0$;
- The projection does not exist.

If the projection $(d_1 + \tau, ..., d_n + \tau)$ exists and $\tau \leq 0$, then d dominates this projection, and the optimisation procedure terminates because the projection is a feasible point. The following lemma considers the second case.

Lemma 4 *If $(d_1 + \tau, ..., d_n + \tau)$ is on the boundary of X and $\tau > 0$, then d does not dominate any feasible point.*

Proof: Since $(d_1 + \tau, ..., d_n + \tau)$ is on the boundary of X, by the definition of the boundary of X, for any $(x_1, ..., x_n) \in X$, there exists j such that $x_j \geq d_j + \tau$, and therefore $\max\limits_{1 \leq i \leq n} (x_i - d_i) \geq \tau$. Hence, $\varrho(d_1, ..., d_n) \geq \tau > 0$ and d does not dominate any feasible point. $\square$.

Observe, that if the projection $(d_1 + \tau, ..., d_n + \tau)$ is on the boundary of X and dominates a, then a does not dominate any feasible point, and entire search tree should be fathomed. Indeed, if we assume that a dominates a feasible point, then the projection dominates the point, which contradicts the assumption that the projection is on the boundary of X.

The two remaining cases are when the projection $(d_1 + \tau, ..., d_n + \tau)$ does not belong to the boundary of X and $\tau > 0$, and the case when the projection does not exist. The lemma below addresses these cases. Let $X(d_1, ..., d_n)$ be the set of all $(x_1, ..., x_n) \in X$ such that

$$\max_{1 \le j \le n} [x_j - d_j] = \varrho(d_1, ..., d_n).$$

Lemma 5 *If $\tau > 0$ and $(d_1 + \tau, ..., d_n + \tau)$ does not belong to the boundary of X or if the projection does not exist, then there exist $(x_1, ..., x_n) \in X(d_1, ..., d_n)$ and i such that*

$$x_i - d_i < \varrho(d_1, ..., d_n). \tag{4.4.3}$$

Proof: Observe that the statement of this theorem does not hold if and only if $X(d_1, ..., d_n)$ is comprised of only point $(d_1 + \varrho(d_1, ..., d_n), ..., d_n + \varrho(d_1, ..., d_n))$. However, in this case for any $(x_1, ..., x_n) \notin X(d_1, ..., d_n)$, $\max_{1 \le j \le n}[x_j - d_j] > \varrho(d_1, ..., d_n)$, and hence there exits i such that $x_i - d_i > \varrho(d_1, ..., d_n) > 0$, which implies that the only point $(d_1 + \varrho(d_1, ..., d_n), ..., d_n + \varrho(d_1, ..., d_n)) \in X(d_1, ..., d_n)$ does not dominate any other feasible point and therefore it is on the boundary of X.

If the projection does not exist, then $(d_1 + t, ..., d_n + t) \notin X$ for all integer t. In particular, $(d_1 + \varrho(d_1, ..., d_n), ..., d_n + \varrho(d_1, ..., d_n))$ is not in X and therefore is not in $X(d_1, ..., d_n)$, because $X(d_1, ..., d_n)$ is a subset of X. Hence, for any $(x_1, ..., x_n) \in X(d_1, ..., d_n)$, there exists i satisfying (4.4.3).

Suppose that the projection exists but $(d_1 + \tau, ..., d_n + \tau) \notin X(d_1, ..., d_n)$. Assume that $(d_1 + \varrho(d_1, ..., d_n), ..., d_n + \varrho(d_1, ..., d_n)) \in X(d_1, ..., d_n)$. Then, by the definition of projection, $\tau < \varrho(d_1, ..., d_n)$, which by virtue of $(d_1 + \tau, ..., d_n + \tau) \in X$ leads to the

following contradiction:

$$\varrho(d_1, ..., d_n) > \tau \geq \min_{(x_1,...,x_n)\in X} \max_{1\leq j\leq n} [x_j - d_j] = \varrho(d_1, ..., d_n).$$

So, $(d_1 + \varrho(d_1, ..., d_n), ..., d_n + \varrho(d_1, ..., d_n)) \notin X(d_1, ..., d_n)$, and therefore for any $(x_1, ..., x_n) \in X(d_1, ..., d_n)$ there exists i satisfying (4.4.3).

Finally, assume that $(d_1 + \tau, ..., d_n + \tau) \in X(d_1, ..., d_n)$. Then, $\tau = \varrho(d_1, ..., d_n)$. Furthermore, since $(d_1 + \tau, ..., d_n + \tau)$ is not on the boundary of X, $(d_1 + \tau, ..., d_n + \tau)$ strictly dominates some $(x_1, ..., x_n) \in X$, i.e. $x_j \leq d_j + \tau$ for all $1 \leq j \leq n$ and at least one of these inequalities is strict. Then, taking into account the definition of $\varrho(d_1, ..., d_n)$,

$$\varrho(d_1, ..., d_n) = \min_{(y_1,...,y_n)\in X} \max_{1\leq j\leq n} [y_j - d_j] \leq \max_{1\leq j\leq n} [x_j - d_j] \leq \tau = \varrho(d_1, ..., d_n).$$

Hence, $(x_1, ..., x_n) \in X(d_1, ..., d_n)$, and (4.4.3) holds for this $(x_1, ..., x_n)$. □

In general, neither index i nor $\varrho(d_1, ..., d_n) - [x_i - d_i]$ are known. The idea is

- find a subset $B \subseteq \{1, ..., n\}$ such that for some $i \in B$, there exists $(x_1, ..., x_n) \in X$, satisfying (4.4.3);
- calculate a lower bound δ: $0 < \delta \leq \varrho(d_1, ..., d_n) - [x_i - d_i]$;
- introduce for each $j \in B$, a new node corresponding to the point $(d'_1, ..., d'_n)$, where
$$d'_e = \begin{cases} d_e - \delta & \text{if } e = j \\ d_e & \text{if } e \neq j \end{cases},$$
if $(d'_1, ..., d'_n)$ is in the domain of F;
- link each such new node with the node corresponding to d.

Though $\{1, ..., n\}$ is always a possible choice of B, smaller B may improve the convergence. It is easy to see that each $(d'_1, ..., d'_n)$ is dominated by d and therefore is dominated by the element of the level set corresponding to the root of the search tree.

Furthermore, for at least one of these new points

$$\varrho(d'_1, ..., d'_n) = \varrho(d_1, ..., d_n).$$

Therefore, if there are feasible points dominated by $(d_1, ..., d_n)$, then at least one of them is dominated by one of the new points. Since $0 \leq a_e \leq p(n)$ for all $1 \leq e \leq n$, after a finite number of steps, the optimisation procedure either finds a feasible point dominated by the considered element of the level set and terminates with this feasible point as an optimal solution, or establishes that the considered element of the level set does not dominate any feasible point.

In general, the methods of choosing B and calculating δ depend on F and X. An example is provided in Section 4.5.

4.4.3 Ascending method

In the ascending method, a search tree for each $a = (a_1, ..., a_n) \in A(\bar{F}, F)$ emanates from the root corresponding to a point that *is dominated* by a and by all points in X. Although $(0, ..., 0)$ is an obvious choice for the point associated with the root, a point with a larger value of the objective function may improve the convergence. As in the descending method, all nodes in this search tree correspond to points in the domain of F and each of these points is dominated by a. In contrast to the descending method, in the ascending method each branching results in new nodes associated with the points that *dominate* the point corresponding to the node at which branching occurred.

Let $b = (b_1, ..., b_n) \notin X$ be a point associated with a node without successors in the partially constructed search tree, and let $(b_1 + \tau, ..., b_n + \tau)$ be the projection of b onto X, if it exists. Consider the following cases:

- The projection exists and the following inequality holds:

$$\tau \leq \min_{1 \leq j \leq n} [a_j - b_j]; \tag{4.4.4}$$

- The projection $(b_1+\tau, ..., b_n+\tau)$ belongs to the boundary of X, and the following inequality holds:

$$\tau \geq \max_{1\leq j\leq n}[a_i - b_i]; \tag{4.4.5}$$

- The projection $(b_1 + \tau, ..., b_n + \tau)$ belongs to the boundary of X, and

$$\min_{1\leq j\leq n}[a_j - b_j] < \tau < \max_{1\leq j\leq n}[a_i - b_i]; \tag{4.4.6}$$

- The projection $(b_1 + \tau, ..., b_n + \tau)$ does not belong to the boundary of X and (4.4.4) does not hold;

- The projection does not exist.

In the first case, since a dominates b, if (4.4.4) holds, then a dominates $(b_1 + \tau, ..., b_n + \tau)$ and the optimisation procedure terminates with the projection as an optimal solution.
The following lemma considers the second case.

Lemma 6 *If the projection $(b_1 + \tau, ..., b_n + \tau)$ is on the boundary of X and (4.4.5) holds, then the point a does not dominate any feasible point.*

Proof: The conditions of the lemma imply that the projection dominates a. If a dominates a feasible point (recall that $a \notin X$), then the projection dominates the point too, which contradicts the assumption that the projection is on the boundary of X. □

As a does not dominate any feasible point in this case, the entire search tree is fathomed. The next lemma addresses the third case.

Lemma 7 *If the projection $(b_1 + \tau, ..., b_n + \tau)$ is on the boundary of X, (4.4.6) holds and there exists $(x_1, ..., x_n) \in X$ dominated by a, then there exists i such that,*

$$x_i - b_i > \min_{1\leq j\leq n}[x_j - b_j]. \tag{4.4.7}$$

Proof: Consider the set of all indexes such that $b_i + \tau + 1 \leq a_i$. Denote this set by B. Observe, that $B \neq \emptyset$ as otherwise (4.4.6) would not hold. Since the projection $(b_1 + \tau, ..., b_n + \tau)$ cannot dominate feasible points, for any feasible point $(x_1, ..., x_n)$ dominated by a (if such a point exists), there exists $i \in B$ satisfying the inequality $x_i \geq b_i + \tau + 1$. The inequality implies that

$$x_i - b_i \geq \tau + 1 > \tau = \min_{1 \leq j \leq n} [x_j - b_j].$$

□

Hence, the node, associated with $(b_1, ..., b_n)$, can be linked with $|B|$ new nodes (branching), one for every $i \in B$. Here a new node corresponding to i is associated with $(b'_1, ..., b'_n)$, where

$$b'_j = \begin{cases} b_j + \tau + 1 & \text{if } j = i, \\ b_j & \text{if } j \neq i. \end{cases}$$

The following lemma considers the two remaining cases: when the projection $(b_1 + \tau, ..., b_n + \tau)$ does not belong to the boundary of X and (4.4.4) does not hold and when the projection does not exist. Let $X(b, a)$ be the set of all feasible points which dominate b and are dominated by a.

Lemma 8 *If the projection $(b_1 + \tau, ..., b_n + \tau)$ is not on the boundary of X and (4.4.4) does not hold or if the projection does not exist, and if $X(b, a) \neq \emptyset$, then for any $(x_1, ..., x_n) \in X(b, a)$, there exists i for which the inequality (4.4.7) holds.*

Proof: Assume that $X(b, a) \neq \emptyset$ and the projection does not exist. If there is $(x_1, ..., x_n) \in X(b, a)$ such that all n differences $x_i - b_i$ are equal and t is their common value, then

$$(x_1, ..., x_n) = (b_1 + t, ..., b_n + t).$$

Hence the projection exists, which contradicts the initial assumption.

Assume that $X(b, a) \neq \emptyset$, the projection is not on the boundary of X and (4.4.4) holds. If there exist $(x_1, ..., x_n) \in X(b, a)$ such that all n differences $x_i - b_i$ are equal

and t is their common value, then

$$t = \min_{1 \le j \le n} [x_j - b_j] \le \min_{1 \le j \le n} [a_j - b_j],$$

which implies that (4.4.4) holds - which again contradicts the initial assumption.

Since for any $(x_1, ..., x_n) \in X(b, a)$ not all differences $x_i - b_i$ are equal, then there exists i such that

$$x_i - b_i > \min_{1 \le j \le n} [x_j - b_j].$$

□

In general, neither $(x_1, ..., x_n) \in X(b, a)$ nor index i are known, and the idea is:

- find a subset $B \subseteq \{1, ..., n\}$ such that if $X(b, a) \neq \emptyset$, there exists $(x_1, ..., x_n) \in X(b, a)$ and $i \in B$ such that (4.4.7) holds;
- calculate a lower bound δ: $1 \le \delta \le x_i - b_i - \min_{1 \le j \le n} [x_j - b_j]$;
- to introduce, for each $j \in B$, a new node corresponding to the point $b' = (b'_1, ..., b'_n)$, where

$$b'_e = \begin{cases} b_e + \delta & \text{if } e = j \\ b_e & \text{if } e \neq j \end{cases},$$

  if b' is dominated by a;
- link each such new node with the node corresponding to b.

Similar to the descending method, $\{1, ..., n\}$ is an obvious choice for B, but smaller B may improve the convergence.

It is easy to see that if $X(b, a) \neq \emptyset$, then for at least one $(x_1, ..., x_n) \in X(b, a)$ there exists a new node and the associated point $b' = (b'_1, ..., b'_n)$ such that

$$\min_{1 \le j \le n} [x_j - b'_j] = \min_{1 \le j \le n} [x_j - b_j] \ge 0.$$

Hence, if $X(b, a) \neq \emptyset$, then $X(b', a) \neq \emptyset$ for at least one new node. On the other hand, each $(b'_1, ..., b'_n)$ dominates b. Since all coordinates of all points are nonnegative

integers bounded above by $p(n)$, after a finite number of steps, the optimisation procedure either finds a feasible point dominated by the considered element of the level set and terminates with this feasible point as an optimal solution, or establishes that the considered element of the level set does not dominate any feasible point.

4.4.4 Descending-ascending method

As the name suggests, the descending-ascending method is a combination of the descending and ascending methods. Therefore, each node of the search tree for the currently considered $a = (a_1, ..., a_n) \in A(\bar{F}, F)$ is associated with a pair of points. All these points are in the domain of F. The root of the search tree is associated with a pair of points where the first point is a point dominated by all points in X, i.e. by all feasible points, and by a, whereas the second point is a itself. For all other nodes of the search tree, the pair (b, d) of points, associated with a node, has the following property: $b \notin X$, $d \notin X$, d dominates b, and a dominates d.

Let b and d be a pair of points, associated with a node of the search tree that does not have successors in this tree. The search procedure attempts to find a feasible point that dominates b and is dominated by a, or to determine that such feasible point does not exist and hence the node has to be fathomed. First procedure applies the ascending method, ignoring d. If the ascending method cannot achieve this goal, the procedure ignores b and uses the descending method to find a feasible point dominated by d.

If both ascending and descending methods, using point b and point d respectively, have failed to find a desired feasible point or to establish that such point does not exist, then branching occurs. Branching is generated either using the ascending method and the point b, or using the descending method and the point d, but not using both methods simultaneously. If the ascending method is selected for branching, this method is modified to satisfy the following condition: each resultant point b' should be dominated by d. If the descending method is selected for branching, this method is modified to satisfy the following condition: each resultant point d' should dominate b. The choice of what method among the two should be used for

branching can be made in many different ways. For example, the selection criterion may be the cardinality of the sets B, generated by each of the methods. To provide another example, let $b = (b_1, ..., b_n)$, $d = (d_1, ..., d_n)$, and $b' = (b'_1, ..., b'_n)$, and $d' = (d'_1, ..., d'_n)$ be the points introduced by branching by ascending and descending methods, correspondingly. Then a selection criterion can be based on the comparison of $\max_{1\leq i\leq n}(d_i - b'_i)$ and $\max_{1\leq i\leq n}(d'_i - b_i)$.

4.5 Application

4.5.1 Description of the problem and preliminaries

As an illustration, consider the following scheduling problem

$$P|prec, c_{ij} = 1, p_i = 1|L_{max}, \tag{4.5.1}$$

where P indicates that a set of tasks $N = \{1, ..., n\}$ is processed on parallel machines, $prec$ and $p_i = 1$ specify that tasks are subject to precedence constraints and that all processing times are one unit of time, $c_{ij} = 1$ signifies the unit communication delays, that is if $i \to j$ and these two tasks are processed on different machines, then j can commence its processing only after completion of i plus an additional unit of time. No preemptions are allowed; a feasible schedule σ is specified by tasks' completion times $C_i(\sigma)$, $1 \leq \ i \leq n$, which are assumed to be positive integers. Let m be the number of parallel machines. It is easy to see that even if the tasks are scheduled one at a time, $C_i(\sigma) \leq n$, $1 \leq i \leq n$. Thus the feasible region X can be viewed as a set of points with integer coordinates $(C_1(\sigma), ..., C_n(\sigma))$ satisfying:

a. $1 \leq C_i(\sigma) \leq n$, $1 \leq \ i \leq n$;

b. $|\{i : \ C_i(\sigma) = t\}| \leq m$ for all integer $1 \leq t \leq n$;

c. $C_i(\sigma) \leq C_j(\sigma) - 1$ for all i and j such that $i \to j$;

d. if $i \to j$ and $i \to g$, then $C_i(\sigma) \leq \max\{C_j(\sigma), C_g(\sigma)\} - 2$;

e. if $i \to j$ and $h \to j$, then $C_j(\sigma) \geq \min\{C_i(\sigma), C_h(\sigma)\} + 2$.

The objective is to minimize maximum lateness

$$L_{max}(\sigma) = \max_{j \in N}(C_j(\sigma) - d_j), \tag{4.5.2}$$

where $C_j(\sigma)$ is the completion time of a task j in schedule σ and d_j is the task's due date, assumed to be integer.

The (4.5.1) scheduling problem possesses the required three properties. Indeed, the objective function of maximum lateness is a particular case of the objective function (4.3.1), discussed in the subsection 4.3.2: for any $i \in N$ $\varphi_i = C_i - d_i$. Hence, the problem possesses *Property 2* and *Property 3*. To show that (4.5.1) possesses *Property 1* we modify Lemma 2 as it is shown below. The conditions of the lemma can be checked in $O(n^2)$, and hence (4.5.1) has *Property 1*.

Lemma 9 *A point $(x_1, ..., x_n) \in X$ is on the boundary of X if and only if, for each integer $t \geq 1$ such that $|\{g : \ x_g = t\}| < m$ and for each $x_j > t$, at least one of the following conditions hold:*

- *there exists i such that $i \to j$ and $x_i \geq t$;*
- *there exist i and k such that $i \to j$, $i \to k$, $x_i = t - 1$ and $x_k = t$;*
- *there exist i and h such that $i \to j$, $h \to j$ and $\min\{x_i, x_h\} \geq t - 1$.*

Proof: Suppose that $x = (x_1, ..., x_n) \in X$ is on the boundary of X, and let $t \geq 1$ be any integer such that $|\{g : \ x_g = t\}| < m$. Let j be any task such that $x_j > t$. Consider the point $x' = (x'_1, ..., x'_n)$, where $x'_j = t$ and $x'_g = x_g$ for all $g \neq j$. Since x is on the boundary of X and x strictly dominates x', $x' \notin X$. Hence, x' does not satisfy some feasibility conditions. The conditions (a.) and (b.) are not violated as $1 \leq t < x_j$, hence $1 \leq x'_i \leq n$ for all $i \in N$; and $|\{g : \ x_g = t\}| < m$ implies that $|\{g : \ x'_g = t\}| \leq m$. Therefore, at least one of the conditions (c.), (d.) or (e.) is not satisfied. Suppose x' does not satisfy the feasibility condition (c.). This implies that there exists $i \to j$ and $x_i \geq t$. If (d.) is not satisfied, then there exists $i \to j$ and k

such that $i \to k$, and $x_i = t-1$, $x_k = t$. Finally, if (e.) is violated, then there exist i and h such that $i \to j$, $h \to j$ and $\min\{x_i, x_h\} \geq t-1$.

Conversely, consider $x = (x_1, ..., x_n) \in X$ and suppose that, for each integer $t \geq 1$ such that $|\{g : \ x_g = t\}| < m$ and each j such that $x_j > t$, either there exists i such that $x_i \geq t$ and $i \to j$, or there exist i and k such that $i \to j$, $i \to k$, $x_i = t-1$ and $x_k = t$; or there exist i and h such that $i \to j$, $h \to j$ and $\min\{x_i, x_h\} \geq t-1$. Suppose that x is not on the boundary of X, i.e. x strictly dominates some $x' = (x'_1, ..., x'_n) \in X$. Then, among all g such that $x'_g < x_g$ select one with the smallest x'_g. Let it be task j. Let $\tau = x'_j$. Then if for some g $x'_g = \tau$, it implies that $x'_g = x_g$. Further $x'_j < x_j$ implies that $|\{g : \ x_g = \tau\}| < m$ and hence one of the conditions of the lemma holds. Consequently, then for the x' one of the feasibility conditions is violated. Indeed, the existence of i such that $x_i \geq \tau$ and $i \to j$ contradicts $x' \in X$ because $x'_i = x_i \geq x'_j$. Further, if there exist i and k such that $i \to j$, $i \to k$, $x_i = \tau - 1 = x'_i$ and $x_k = \tau = x'_k$, then again $x' \notin X$ because

$$x'_i = \tau - 1 > \tau - 2 = \max\{x_j, x_k\} - 2.$$

Finally, if there exist i and h such that $i \to j$, $h \to j$ and $\min\{x_i, x_h\} \geq \tau - 1$, then

$$x'_j = \tau < \tau + 1 \leq \min\{x_i, x_h\} + 2 = \min\{x'_i, x'_h\} + 2.$$

□

In what follows the time slot t is an interval $[t-1, t]$. The time slot t is complete (in respect to the given priority μ_g) if there are m tasks j with priority $\mu_j \geq \mu_g$ and completion time $C_j(\sigma) = t$, otherwise the time slot is incomplete. Let $Q(i)$ be the set of predecessors of task i and $K(i)$ be the set of successors of the task i, correspondingly. Then for each task $i \in N$ calculate the following:

- The lower bound for completion time c_i:

$$c_i = \begin{cases} 1, & \text{if } Q(i) = \emptyset \\ \max\limits_{\underline{c} \leq c \leq \bar{c}} \left\{ c + \left\lceil \dfrac{|\{j : j \in Q(i) \text{ and } c_j \geq c\}| - 1}{m} \right\rceil + 1 \right\} & \text{otherwise,} \end{cases} \tag{4.5.3}$$

where $\underline{c}$ and $\bar{c}$ are the smallest and the largest c_j among $j \in Q(i)$.

- The priority μ_i:

$$\mu_i = \begin{cases} \max\limits_{j \in N} d_j - d_i, & \text{if } K(i) = \emptyset \\ \max\left\{ \max\limits_{j \in N} d_j - d_i, \max\limits_{\underline{\mu} \le \mu \le \bar{\mu}} \left\{ \mu + \left\lceil \frac{|\{j : j \in K(i) \text{ and } \mu_j \ge \mu\}| - 1}{m} \right\rceil + 1 \right\} \right\} & \text{otherwise,} \end{cases} \tag{4.5.4}$$

where $\underline{\mu}$ and $\bar{\mu}$ are the smallest and the largest μ_j among $j \in K(i)$, and d_i is the task's due date.

It has been shown in [113], that for any schedule σ

$$L_{max}(\sigma) = \max_{i \in N}(C_i(\sigma) + \mu_i) - \max_{j \in N} d_j. \tag{4.5.5}$$

According to the optimisation procedure, the first step is to calculate the initial lower bound $\underline{L}$. Taking into account (4.5.5), we have

$$\underline{L} = \underline{G} - \max_{q \in N} d_q, \tag{4.5.6}$$

where $\underline{G}$ is the lower bound on $\max\limits_{i \in N}(C_i(\sigma) + \mu_i)$. The next step of the procedure requires to determine a level set for this value of the objective function. The level set consists of only one point $(a_1, ..., a_n)$ for any value $\underline{L}$ of the maximum lateness function, where a_i, for $1 \le i \le n$, is defined as $a_i = \underline{L} + d_i$.

Thus, for the only point of the level set $(a_1, ..., a_n)$, the procedure determines whether or not there exists a feasible point $(x_1, ..., x_n) \in X$, which is dominated by $(a_1, ..., a_n)$. In other words, is there a feasible schedule σ, such that

$$\max_{i \in N}(C_i(\sigma) - a_i) \le 0 \tag{4.5.7}$$

Taking into account (4.5.5), (4.5.6) and the selection of a_i,

$$\max_{i \in N}(C_i(\sigma) - a_i) = \max_{i \in N}(C_i(\sigma) - (\underline{L} + d_i)) =$$
$$\max_{i \in N}(C_i(\sigma) + \mu_i) - \max_{q \in N} d_q - (\underline{G} - \max_{q \in N} d_q) = \max_{i \in N}(C_i(\sigma) + \mu_i) - \underline{G}.$$

Thus the inequality (4.5.7) holds if and only if the following inequality holds:

$$\max_{i\in N}(C_i(\sigma)+\mu_i)\le \underline{G}. \tag{4.5.8}$$

Observe that due to feasibility condition (b.), for any $V\subseteq N$,

$$\min_{i\in V} C_i(\sigma)+\frac{|V|}{m}-1\le \min_{i\in V} C_i(\sigma),$$

hence the lower bound $\underline{G}$ is calculated as follows:

$$\underline{G}=\max_{\underline{c}\le c\le \bar{c}}\left\{\max_{\underline{\mu}\le\mu\le\bar{\mu}}\left\{c-1+\left\lceil\frac{|\{j: c_j\ge c \text{ and } \mu_j\ge\mu\}|}{m}\right\rceil+\mu\right\}\right\}, \tag{4.5.9}$$

where $\underline{c}$ and $\bar{c}$ are the smallest and the largest c_i among $i\in N$ and $\underline{\mu}$ and $\bar{\mu}$ are the smallest and the largest μ_i among $i\in N$.

In order to find a feasible schedule satisfying (4.5.8) for the current value of $\underline{G}$, the procedure constructs a search tree using descending, ascending, or descending-ascending methods.

Let $b=(b_1,...,b_n)$ be a point associated with a node of a partially constructed search tree. If $b\in X$ then the procedure terminates with an optimal schedule σ such that $C_i(\sigma)=b_i$ for any $i\in N$. Assume that $b\notin X$. Hence at least one of the feasibility conditions (a.)-(e.) does not hold. Then for any integer t the point $(b_1+t,....,b_n+t)\notin X$ either, which implies that the projection of b onto X does not exist.

If the procedure determines that such a schedule does not exist, it retains the value $\tilde{G}$, the next smallest value of $\max_{i\in N}(C_i(\sigma)+\mu_i)$ such that $\tilde{G}>\underline{G}$. The new iteration starts with the new lower bound $\tilde{G}>\underline{G}$. Once a schedule satisfying (4.5.8) for the current value of $\underline{G}$ is found, the optimal value L^* is calculated as $L^*=\underline{G}-\max_{q\in N} d_q$.

4.5.2 Descending method

According to the descending method described in subsection 4.4.2 the root of the search tree is associated some point $a\in A(\bar{F},F)$ and each node of the tree is associ-

ated with some point d which is dominated by a. Taking into account (4.5.5), each node of the search tree for the descending method, applied to the (4.5.1) problem, is associated with the set of priorities $\{\mu_1, ..., \mu_n\}$. Further, as no point corresponding to a node in the partially constructed search tree, is a feasible point, the descending method calculates lower bound for the current set of priorities $\{\mu_1, ..., \mu_n\}$, and either finds an optimal schedule, or branches on the current node, or determines that no further branching is required.

The descending method will utilise the *Descending* list algorithm, which can be described as follows. Assume N' consists of all tasks of N numbered in non-increasing order of priorities $\{\mu_1, ..., \mu_n\}$, plus there is the last additional unit task $n+1$. Let $D(t)$ be a subset of tasks $j \in N'$ for which a completion time has not been assigned yet, and either $Q(j) = \emptyset$ or $j \in D(t)$ possesses the following properties:

- $C_j(\sigma)$ has not been assigned;
- for all $v \in Q(j)$ the completion time has been assigned and $C_v(\sigma) < t$;
- $|\{v \in Q(j) : \ C_v(\sigma) = t-1\}| \leq 1$;
- for any $v \in Q(j) : \ C_v(\sigma) = t-1$, $K(v) \cap \{h \in N' : \ C_h(\sigma) = t\} = \emptyset$

Descending list algorithm

Set $t = 1$, $N_{sch} = 0$, $n_m = 0$, $i = 0$;
while $N_{sch} < N$ **do**
Set $k = \min\{v > i : \ v \in D(t)\}$;
if $k < n+1$ **and** $n_m < m$ **then**
Set $i = k$, $C_k(\sigma) = t$; $N_{sch} = N_{sch} + 1$; $n_m = n_m + 1$;
else
Set $t = t+1$, $i = 0$, $n_m = 0$.
end if
end while

The Descending list algorithm constructs a feasible schedule:

- the condition [a.] is satisfied as for any value of $t \geq 1$ either there is at least one task is assigned a completion time or in a previous time slot there are at least two tasks scheduled;
- the operator **IF** guarantees that the condition [b.] is observed;
- the selection of the set $D(t)$ and index k guarantee conditions [c.]-[e.].

Let σ be the schedule constructed by the Descending list algorithm in non-increasing order of the set of priorities μ, associated with the current node. Denote by $G_{max}(\sigma) = \max\limits_{i \in N}(C_i(\sigma) + \mu_i)$. It has been shown in [113] that optimality of σ and the branching set B can be determined as follows:

- Select the task g with the smallest completion time $C_g(\sigma)$ among all tasks j such that $C_j(\sigma) + \mu_j = G_{max}(\sigma)$, and let τ be the first incomplete time slot (in respect to the priority μ_g) on the left of the time slot $\tau = C_g(\sigma)$ in the schedule σ.
- If $\tau = 1$, or there are no incomplete time slots on the left of the task g, then σ is the optimal schedule for this set of priorities μ.
- If $\tau > 1$, then select the following sets:

$$U = \{u : \ \tau < C_u(\sigma) < C_g(\sigma) \text{ and } \mu_u \geq \mu_g\} \cup \{g\};$$
$$T = \{b : \overline{K(b)} \cap U \neq \emptyset \text{ and } C_b(\sigma) = \tau - 1\};$$
$$S = \{b : K(b) \cap U \neq \emptyset, Q(b) \cap T = \emptyset \text{ and } C_b(\sigma) = \tau\};$$

  where $\overline{K(b)}$ is the set of immediate successors of task b.
- Set $B = T \bigcup S$. It has been shown in [113] that there exists an optimal schedule σ^* and $j \in B$ such that $G_{max}(\sigma^*) > C_j(\sigma^*) + \mu_j$ and thus the priority μ_j can be increased by $\delta = C_g(\sigma) + \mu_g - (C_j(\sigma) + \mu_j)$ without changing the value of the criterion.

Consider the partially constructed search tree associated with the current lower bound $\underline{G}$. The lower bound LB^{μ} is calculated with the set of priorities according to (4.5.9). All current existing nodes are queued in the list in non-decreasing order of corresponding lower bounds. Denote by σ^f the schedule with the smallest value G^f produced by a feasible schedule so far. Let $\tilde{G} \leq G^f$ be the smallest lower bound on the optimal value of G_{max} the tree has produced so far. Here are the steps of the descending method for the current lower bound $\underline{G}$:

1. Select the node with the smallest lower bound LB^{μ}. If the list is empty and $\tilde{G} > \underline{G}$, the new iteration of the optimisation procedure starts for the new lower bound $\tilde{G}$.

2. Construct a schedule σ with *Descending* list algorithm and the list of tasks in non-increasing order of μ.

3. If $G_{max}(\sigma) \leq \underline{G}$, set $L^* = \underline{G} - \max\limits_{q \in N} d_q$ and terminate the optimisation procedure. Otherwise go to the next step.

4. Select the task g and τ as described above. If $\tau = 1$, or there are no incomplete time slots on the left of g, then σ is the optimal schedule for this set of priorities μ. If $G_{max}(\sigma) < \tilde{G}$, set $\tilde{G} = C_{max}(\sigma)$. If $G_{max}(\sigma) < G^f$, set $G^f = C_{max}(\sigma)$ and $\sigma^f = \sigma$. No further branching is required on the node, delete the node and go to the step 1. If $\tau > 1$ go to the next step.

5. Select sets U, T and S; set $B = T \bigcup S$.

6. For each $i \in B$ calculate the new set of the priorities η as follows:

$$\eta_j = \begin{cases} \mu_j, & \text{if } j \neq i, \\ \eta_j = \mu_j + C_g(\sigma) + \mu_g - (C_j(\sigma) + \mu_j), & \text{if } j = i. \end{cases}$$

7. Calculate lower bound LB^{η}. If $LB^{\eta} \leq \underline{G}$, create a new node associated with the set of priorities η and the lower bound LB^{η}. Include the node in the list of nodes and re-arrange the list in non-decreasing order of nodes' lower bounds. If $\underline{G} < LB^{\eta} < \tilde{G}$, let $\tilde{G} = LB^{\eta}$.

8. If all elements of the set B have been considered, delete the current node and go to step 1.

4.5.3 Ascending method

The main difference of the ascending method from the descending method is that we will be increasing and branching on the sets of the lower bounds $\{c_1, ..., c_n\}$ instead of increasing and branching on the sets of priorities $\{\mu_1, ..., \mu_n\}$. Let $\mu^1 < \mu^2 < ... < \mu^r$ be all different values of the priorities μ, and n^i be the number of tasks with μ^i, $1 \leq i \leq r$. Similar to the reasoning in section 4.3.1 regarding values of release times it is assumed that

$$\mu^1 + \sum_{i=1}^{e} n^i > \mu^{e+1} \tag{4.5.10}$$

for any $1 \leq e \leq r$. Indeed, if (4.5.10) does not hold for some e, then the problem can be split into two subproblems: first, all tasks with $\mu^1, ..., \mu^e$ are to be scheduled, and then the remaining tasks are to be scheduled.

To select the branching set, a schedule σ is constructed using *Ascending* list algorithm with the list of tasks in non-increasing order of c. The schedule is constructed from right to left starting from the time slot $t = n$. Assume N' consists of all tasks of N numbered in non-increasing order of priority c plus the last additional unit task $n+1$. Let $A(t)$ be a subset of tasks $j \in N'$ for which a completion time has not been assigned yet, and either $K(j) = \emptyset$ or $j \in K(t)$ possesses the following properties:

- $C_j(\sigma)$ has not been assigned;
- $t + \mu_j \leq n$;
- for all $v \in K(j)$ the completion time has been assigned and $C_v(\sigma) > t$;
- $|\{v \in K(j) : \ C_v(\sigma) = t + 1\}| \leq 1$;
- for any $v \in K(j) : \ C_v(\sigma) = t + 1$, $Q(v) \cap \{h \in N' : \ C_h(\sigma) = t\} = \emptyset$

Ascending list algorithm

Set $t = n$, $N_{sch} = 0$, $n_m = 0$, $i = 0$;

while $N_{sch} < N$ **do**

Set $k = \min\{v > i : \ \ v \in A(t)\}$;

if $k < n+1$ **and** $n_m < m$ **then**

Set $i = k$, $C_k(\sigma) = t$; $N_{sch} = N_{sch} + 1$; $n_m = n_m + 1$;

else

Set $t = t - 1$, $i = 0$, $n_m = 0$

end if

end while

The *Ascending* list algorithm constructs a feasible schedule:

- the condition [a.] is satisfied as for any value of $t < n$ there is at least one task is assigned a completion time or there is at least two tasks into a previous time slot $t + 1$;
- the operator **IF** guarantees that the condition [b.] is observed;
- the selection of the set $A(t)$ and index k guarantee conditions [c.]-[e.].

It is easy to see that by construction

$$\max_{i \in N}(C_i(\sigma) + \mu_i) = n. \tag{4.5.11}$$

Let $\chi = \min_{j \in N}(C_j(\sigma) - c_j)$. Let σ' be a schedule derived from σ by shifting all completion times by χ to the left: $C_i(\sigma') = C_i(\sigma) - \chi, 1 \leq i \leq n$. Taking into account (4.5.11) and the definition of χ,

$$\begin{aligned} &\max_{i \in N}(C_i(\sigma') + \mu_i) \leq \underline{G} \Longleftrightarrow \max_{i \in N}(C_i(\sigma) + \mu_i - \chi) \leq \underline{G} \Longleftrightarrow \\ &n - \min_{j \in N}(C_j(\sigma) - c_j) \leq \underline{G} \Longleftrightarrow \min_{j \in N}(C_j(\sigma) - c_j) \geq n - \underline{G}. \end{aligned} \tag{4.5.12}$$

Let σ be the schedule constructed by the Ascending list algorithm in non-increasing order of the set of priorities $\{c_1, ..., c_n\}$, associated with the current node. Denote by

$\chi(\sigma) = \min_{i \in N}(C_i(\sigma) - c_i)$. The optimality of σ in respect to the criterion

$$\max_{(C_1(\sigma'),\ldots C_n(\sigma')) \in X} \min_{i \in N}(C_i(\sigma') - c_i)$$

and the branching set B can be determined as follows:

1. Select the task g with the largest completion time $C_g(\sigma)$ among all tasks j such that $C_j(\sigma) - c_j = \chi(\sigma)$ and let τ be the largest incomplete time slot (in respect to the priority c_g) on the right of g in schedule σ.

2. If $\tau = n$, or there are no incomplete time slots on the right of g, then σ is the optimal schedule for this set of priorities $\{c_1, ..., c_n\}$. This item is addressed in the lemma below.

3. If $\tau < n$, then select the following sets:

$$U = \{u : \ C_g(\sigma) < C_u(\sigma) < \tau \text{ and } c_u \geq c_g\} \cup \{g\};$$
$$T = \{b : \overline{Q(b)} \cap U \neq \emptyset \text{ and } C_b(\sigma) = \tau + 1\};$$
$$S = \{b : Q(b) \cap U \neq \emptyset, K(b) \cap T = \emptyset \text{ and } C_b(\sigma) = \tau\};$$

   where $\overline{Q(b)}$ is the set of immediate predecessors of task b.

4. Set $B = T \bigcup S$. It will be shown that there exists an optimal schedule σ^* and $j \in B$ such that

$$\chi_{min}(\sigma^*) < C_j(\sigma^*) - c_j$$

   and thus the priority c_j can be increased by

$$\delta = C_g(\sigma) - c_g - (C_j(\sigma) - c_j)$$

   without changing the value of the criterion.

Lemma 10 *Let σ be the schedule constructed by the Ascending list algorithm in non-increasing order of the set of priorities $\{c_1, ..., c_n\}$, and let task g be selected as described above in item [1.]. If there are no incomplete time slots (in respect to the*

priority c_g) on the right of g in schedule σ, or the largest incomplete time slot $\tau = n$, then σ is the optimal schedule for this set of priorities $\{c_1, ..., c_n\}$.

Proof: Assume that there are no incomplete time slots on the right of g in σ. Let σ^c be an optimal schedule for this set of priorities c, constructed from right to a left starting from $t = n$. Then

$$\chi(\sigma^c) \leq \min_{i \in U}(C_i(\sigma^c - c_i) \leq \min_{i \in U} C_i(\sigma^c) - \min_{i \in U} c_i \leq n - \left\lceil \frac{U}{m} \right\rceil - c_g$$
$$= C_g(\sigma) - c_g = \chi(\sigma) \leq \max_{(C_1(\sigma'),...C_n(\sigma')) \in X} \min_{i \in N}(C_i(\sigma') - c_i) = \chi(\sigma^c),$$

hence σ is optimal.

Assume that $\tau = n$ is the largest incomplete time slot on the right of g in σ. Denote by B the following set: $B = \{j \in B : \ C_j(\sigma) = n \text{ and } Q(j) \bigcap U \neq \emptyset\}$. It is easy to see that $B \neq \emptyset$ as otherwise $C_i(\sigma) < n$ for all $i \in U$ contradicts the Ascending list algorithm. Hence

$$\chi(\sigma^c) \leq \min_{i \in U}(C_i(\sigma^c - c_i) \leq \min_{i \in U} C_i(\sigma^c) - \min_{i \in U} c_i \leq n - 1 - \left\lceil \frac{U}{m} \right\rceil - c_g$$
$$= C_g(\sigma) - c_g = \chi(\sigma) \leq \max_{(C_1(\sigma'),...C_n(\sigma')) \in X} \min_{i \in N}(C_i(\sigma') - c_i) = \chi(\sigma^c),$$

hence σ is optimal.□

Lemma 11 *Let σ be the schedule constructed by Ascending list algorithm in non-increasing order of the set of priorities $\{c_1, ..., c_n\}$, and let task g be selected as described above in item [1.], and sets U, T, S - as described above in item [3.] and $B = T \bigcup S$. If the largest incomplete time slot (in respect to priority c_g) $\tau < n$, then there exist an optimal schedule σ^* and $j \in B$ such that*

$$\chi(\sigma^*) < C_j(\sigma^*) - c_j$$

and thus the priority c_j can be increased by

$$\delta = C_g(\sigma) - c_g - (C_j(\sigma) - c_j)$$

without changing the value of the criterion.

Proof: Let $q \in B$ be the task with $C_q(\sigma^*) = \max_{i\in B} C_i(\sigma^*)$, and assume that $C_q(\sigma^*) = \max_{i\in S} C_i(\sigma^*)$. Then:

$$\begin{aligned}
&\chi(\sigma^*) \leq \min_{i\in U}(C_i(\sigma^* - c_i) \leq \min_{i\in U} C_i(\sigma^*) - \min_{i\in U} c_i \\
&\leq \max_{i\in B} C_i(\sigma^*) - \left\lceil \frac{|U|}{m} \right\rceil - c_g = C_q(\sigma^*) - (\tau - C_g(\sigma)) - c_g \\
&= C_q(\sigma^*) - (C_q(\sigma) - C_g(\sigma)) - c_g < C_q(\sigma^*) - c_q
\end{aligned} \tag{4.5.13}$$

Now assume that

$$C_q(\sigma^*) > \max_{i\in S} C_i(\sigma^*) \text{ or } S = \emptyset. \tag{4.5.14}$$

Define the following sets:

- $H = \{h \in H : \ C_h(\sigma) = \tau, \ \ K(h) \cap T \neq \emptyset, \ \ c_h \geq c_g\}$;
- $S' = \{s \in S' : \ C_s(\sigma) = \tau, \ \ Q(s) \cap U \neq \emptyset\}$.
- $T' = S' \cup H$;
- $W = \{w \in T' \cup U \text{ and } C_w(\sigma^*) = C_q(\sigma^*) - 1\}$.

Observe, that for any $s \in S'$ and $u \in U$ such that $u \to s$, $c_s \geq c_u \geq c_g$. To show that $|W| \leq |T'|$, similar to [113], the one-to-one mapping of set W onto a subset of T' is defined as follows: for each $w \in W$ an element $w(T') \in T'$ is specified. If $w \in T'$, then $w(T') = w$. Assume that $w \notin T'$. Then by definition of W, $w \in U$. Therefore there exists $b \in B$ such that $w \to b$. Further, since $C_w(\sigma^*) = C_q(\sigma^*) - 1$, and due to selection of q, $C_b(\sigma^*) = C_q(\sigma^*)$. In addition, by virtue of (4.5.14), $C_b(\sigma) = \tau + 1$, hence there exists $h \in H$ such that $h \to b$, hence $w(T') = h$. In summary,

$$|U \cup T' - W| \geq |U \cup T'| - |W| \geq |U \cup T'| - |T'| = |U| + |T'| - |T'| = |U|. \tag{4.5.15}$$

Finally, taking into account (4.5.15),

$$\begin{aligned}
&\chi(\sigma^*) \le \min_{i\in U}(C_i(\sigma^*) - c_i) \\
&\le \min_{i\in U} C_i(\sigma^*) - \min_{i\in U} c_i \le \max_{i\in B} C_i(\sigma^*) - 1 - \left\lceil \frac{|U \cup T' - W|}{m} \right\rceil - c_g \\
&\le C_q(\sigma^*) - 1 - \left\lceil \frac{|U|}{m} \right\rceil - c_g = C_q(\sigma^*) - 1 - (\tau - C_g(\sigma)) - c_g \\
&= C_q(\sigma^*) - (C_q(\sigma) - C_g(\sigma)) - c_g < C_q(\sigma^*) - c_q. \qquad (4.5.16)
\end{aligned}$$

By virtue of (4.5.13) and (4.5.16), there exists $q \in B$ and optimal schedule σ^* such that $C_q(\sigma^*) - c_q > \chi(\sigma^*)$, hence c_q can be increased by $\delta = C_g(\sigma) - c_g - (C_q(\sigma) - c_q)$.□

As shown above, if the schedule σ satisfies (4.5.12), then the corresponding σ' satisfies (4.5.8). Consider the partially constructed search tree associated with the current lower bound $\underline{G}$. Each node of the search tree is associated with the set of lower bounds on the tasks' completion times $\{c_1, ..., c_n\}$ and the lower bound LB^c, where LB^c is calculated with the set of $\{c_1, ..., c_n\}$ according to (4.5.9). All current existing nodes are queued in the list in non-decreasing order of corresponding lower bounds LB^c. Denote by $\chi(\sigma) = \min_{j\in N}(C_j(\sigma) - c_j)$. Denote by σ^f the schedule with the largest value χ^f produced by a feasible schedule so far. Let $\tilde{\chi} \ge \chi^f$ be the largest value the search tree has produced so far. Here are the steps of the ascending method:

1. Select the node with the smallest lower bound LB^c. If the list is empty and $\tilde{\chi} < n - \underline{G}$, the new iteration of the optimisation procedure starts for the new lower bound $\tilde{G} = n - \tilde{\chi}$.

2. Construct a schedule σ using the *Ascending* list algorithm and the list of tasks in non-increasing order of cs.

3. If $\chi(\sigma) \ge n - \underline{G}$, set $L^* = \underline{G} - \max_{q\in N} d_q$ and terminate the optimisation procedure. Otherwise go to the next step.

4. Select task g with the largest completion time $C_g(\sigma)$ among all tasks j such that $C_j(\sigma) - c_j = \chi(\sigma)$ and let τ be the first incomplete time slot (in respect to the priority c_g) on the right of g in schedule σ. If $\tau = n$, or there are no

incomplete time slots on the right of g, then by virtue of lemma 10 σ is the optimal schedule for this set of $\{c_1, ..., c_n\}$. If $\chi(\sigma) > \chi^f$, set $\chi^f = \chi(\sigma)$ and $\sigma^f = \sigma$. If $\chi(\sigma) > \tilde{\chi}$, set $\tilde{\chi} = \chi(\sigma)$. No further branching is required on the node, delete the node and go to the step 1. If $\tau < n$ go to the next step.

5. Select set $U = \{u : \ C_g(\sigma) < C_u(\sigma) < \tau \text{ and } c_u \geq c_g\} \cup \{g\}$ and the sets:

   $T = \{b : \bar{Q(b)} \cap U \neq \emptyset \text{ and } C_b(\sigma) = \tau + 1\}$ and

   $S = \{b : Q(b) \cap U \neq \emptyset, K(b) \cap T = \emptyset \text{ and } C_b(\sigma) = \tau\}$.

   Select the branching set $B = T \bigcup S$. By virtue of lemma 11, there exist an optimal schedule σ^* and $j \in B$ such that $\chi(\sigma^*) < C_j(\sigma^*) - c_j$ and thus the c_j can be increased without changing the value of the criterion.

6. For each $i \in B$ calculate the new set of the priorities ς as follows:

$$\varsigma_j = \begin{cases} c_j, & \text{if } j \neq i, \\ \varsigma_j = c_j + C_j(\sigma) - c_j - (C_g(\sigma) - c_g), & \text{if } j = i. \end{cases}$$

   Calculate lower bound LB^{ς}. If $LB^{\varsigma} \leq \underline{G}$, create a new node associated with the set of priorities ς and the lower bound LB^{ς}. Include the node in the list of nodes and re-arrange the list in non-decreasing order of nodes' lower bounds. If $\underline{G} < LB^{\varsigma} < n - \tilde{\chi}$, let $\tilde{\chi} = n - LB^{\varsigma}$. If all elements of the set B have been considered, delete the current node and go to step 1.

4.5.4 Descending-ascending method

The descending-ascending method incorporates the elements of both descending and ascending methods. Consider the partially constructed search tree associated with the current lower bound $\underline{G}$. Each node of the search tree is associated with the sets $\{c_1, ..., c_n\}$ and $\{\mu_1, ..., \mu_n\}$, and the lower bound $LB^{(c,\mu)}$, where $LB^{(c,\mu)}$ is calculated according to (4.5.9). All current existing nodes are queued in the list in non-decreasing order of corresponding lower bounds $LB^{(c,\mu)}$. Denote by σ^d the schedule constructed by *Descending* list algorithm, and by σ^a- the schedule constructed by *Ascending* list

algorithm. Denote by $\triangle^{(c,\mu)} = \min\{n - \chi(\sigma^a), G_{max}(\sigma^d)\}$. Let $\tilde{\triangle}$ be the smallest value of $\triangle^{(c,\mu)}$ produced by the search tree so far and σ^f be the schedule that with the largest value $\triangle^f \geq \tilde{\triangle}$ produced by a feasible schedule. Here are the steps of the descending-ascending method:

1. Select the node with the smallest lower bound $LB^{(c,\mu)}$. If the list is empty and $\tilde{\triangle} > \underline{G}$, the new iteration starts for the new lower bound $\tilde{G} = \tilde{\triangle}$.

2. Construct a schedule σ^d with the Descending list algorithm and the list of tasks in non-increasing order of μs and a schedule σ^a with the Ascending list algorithm and the list of tasks in non-increasing order of cs.

3. If $\chi(\sigma^a) \geq n - \underline{G}$ or $G_{max}(\sigma^d) \leq \underline{G}$, set $L^* = \underline{G} - \max\limits_{q \in N} d_q$ and terminate the optimisation procedure. Otherwise go to the next step.

4. Select task g^d and incomplete time slot τ^d for the σ^d similar to the step 4 of descending algorithm and select task g^a and incomplete time slot τ^a for the σ^a similar to the step 4 of ascending algorithm. If $\tau^a = n$ or $\tau^d = 1$ or there is no incomplete time slots, then either σ^a or σ^d is the optimal schedule for this set of $\{c_1, ..., c_n\}$ and $\{\mu_1, ..., \mu_n\}$. If $\triangle^{(c,\mu)} < \tilde{\triangle}$, set $\tilde{\triangle} = \triangle^{(c,\mu)}$; if $\triangle^{(c,\mu)} < \triangle^f$, set $\triangle^f = \triangle^{(c,\mu)}$ and $\sigma^f = \sigma$. No further branching is required on the node, delete the node and go to the step 1. If $\tau^a < n$ and $\tau^d > 1$, go to the next step.

5. Select the "descending" and "ascending" branching sets B^d and B^a, repeating the steps 5 and 6 of the descending and ascending methods.

6. Choose the branching set B out of B^d and B^a according to some criterion. Three criteria of selection of the set B are implemented:

   - **Method DA-setB**: Choose the set of the smallest cardinality;
   - **Method DA-MaxWind**: For each element $h \in B^d$ calculate the new priorities $\{\mu_1^h, ..., \mu_n^h\}$ and for each element $q \in B^a$ calculate the new set $\{c_1^q, ..., c_n^q\}$ according to step 6 of the descending and ascending methods

correspondingly. For each branching set calculate the maximum "window":

$$W(B^a) = \max_{j \in B^a}\{\max_{i \in N}(\underline{G} - c_i^j - \mu_i)\};$$
$$W(B^d) = \max_{j \in B^d}\{\max_{i \in N}(\underline{G} - c_i - \mu_i^j)\}.$$

Choose the branching set with the smallest maximum window;

- **Method DA-AveWind**: Similar to the previous case, calculate the set of new priorities for B^d and B^a. Then for each of the sets calculate the average "window":

$$AV(B^a) = \frac{\sum_{j \in B^a}\sum_{i \in N}(\underline{G} - c_i^j - \mu_i)}{|B^a|};$$
$$AV(B^d) = \frac{\sum_{j \in B^d}\sum_{i \in N}(\underline{G} - c_i - \mu_i^j)}{|B^d|};$$

Choose the branching set with the smallest average window.

7. Calculate lower bound $LB^{(c,\mu)}$ for the new sets of cs and μs. If $LB^{(c,\mu)} \leq \underline{G}$, create a new node associated with the sets of priorities and the lower bound $LB^{(c,\mu)}$. Include the node in the list of nodes and re-arrange the list in non-decreasing order of nodes' lower bounds. If $\underline{G} < LB^{(c,\mu)} < \tilde{\triangle}$, let $\tilde{\triangle} = LB^{(c,\mu)}$. If all elements of the set B be have been considered, delete the current node and go to step 1.

4.6 Computational Experiments

The computational experiments aimed to compare the descending, ascending and descending-ascending algorithms. The partially ordered sets were obtained from http://www.kasahara.cs.waseda.ac.jp/schedule/, the Kasahara Laboratory web site. It was assumed that all tasks have unit processing time and there are unit communication delays. Each of the three considered groups consisted of 180 instances of 50, 100 or 300 tasks. For each group the experiments were run for 3, 4, 5 and 6

parallel machines. The execution of each instance was terminated once the search went through 5000 nodes, and the time for each group was limited by 2 hours. Each group is described as $n - m$ and consists of 180 instances each with n tasks and m parallel machines, $n = 50,\ 100,\ 300$; $m = 3,\ 4,\ 5,\ 6$.

Table 4.1 represents the percentage of the instances solved to optimality in each group. All algorithms were effective - across all groups with 50-300 tasks and 3-6 machines every algorithm solved $92 - 100$ percent of the instances to optimality - the schedule was determined as optimal if the value of the objective function was equal to the current lower bound. At least one of the descending-ascending algorithms solved to optimality at least as many or more instances than descending and ascending algorithms. The table shows that there is no difference in performance of DA-MaxWind and DA-AveWind algorithms. Further, the two algorithms provided the same best objective function value for all instances.

Table 4.2 compares descending and ascending methods. The number of iterations required by each method to solve an instance to optimality was calculated, and then the difference between the two numbers was taken. The Table 4.2 shows the percentage of instances where this difference falls into one of the following categories:

- for majority of instances $(78\% - 96\%)$ the numbers of iterations are equal across all groups and numbers of machines;
- for $1.1\% - 11.7\%$ of instances there is a difference within 10 iterations;
- for up to 4.4% of instances there is a difference within 50 iterations;
- for up to 2.2% of instances there is a difference within 100 iterations;
- for up to 3.3% of instances there is a difference within 2000 iterations;
- for up to 1.7% of instances there is a difference within 4500 iterations;
- for up to 2.8% of instances there is a difference of more than 4500 iterations;

which indicates that the algorithms perform differently on the same instances and are sensitive to the structure of the tasks' precedence graphs.

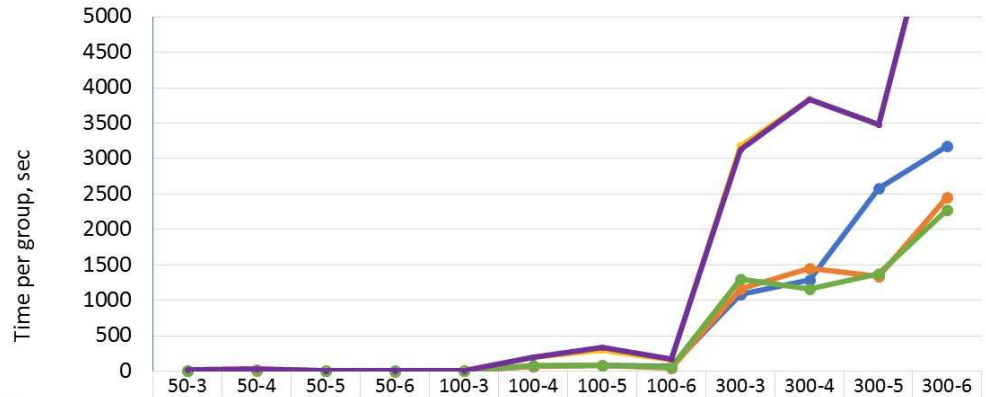
Table 4.1: Proportion of instances, in %, solved to optimality

Group	Ascending	Descending	DA-setB	DA-MaxWind	DA-AveWind
50-3	97.22	**98.33**	95.56	**98.33**	**98.33**
50-4	94.44	**95.56**	93.89	**95.56**	**95.56**
50-5	95.56	**98.33**	**98.33**	**98.33**	**98.33**
50-6	97.78	**99.44**	98.33	**99.44**	**99.44**
100-3	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**
100-4	94.44	**95.56**	93.89	**95.56**	**95.56**
100-5	**96.11**	95.56	95.56	95.56	95.56
100-6	96.67	**97.22**	96.11	**97.22**	**97.22**
300-3	**98.33**	97.78	**98.33**	97.78	97.78
300-4	97.22	97.22	**98.33**	97.22	97.22
300-5	95.00	96.67	**97.22**	**97.22**	**97.22**
300-6	92.22	94.44	**95.56**	95.00	95.00

Table 4.2: Comparison of ascending and descending algorithms: number of iterations

Group	no difference	±10 it	±50 it	±100 it	±2000 it	±4500 it	> 4500 it
50-3	78.3	10.0	4.4	1.1	3.3	1.1	1.7
50-4	78.9	11.7	2.2	1.7	2.2	1.7	1.7
50-5	79.4	11.1	3.9	1.1	1.7	0.0	2.8
50-6	83.9	10.6	2.2	0.6	0.6	1.1	1.1
100-3	94.4	1.1	1.1	2.2	1.1	0.0	0.0
100-4	78.9	11.7	2.2	1.7	2.2	1.7	1.7
100-5	85.0	6.1	3.9	1.1	3.3	0.0	0.6
100-6	85.0	7.8	2.8	2.2	1.1	0.6	0.6
300-3	96.7	1.1	0.6	0.0	0.0	1.7	0.0
300-4	93.9	3.3	0.0	0.0	0.6	0.6	1.7
300-5	92.2	2.2	1.1	0.0	1.1	1.1	2.2
300-6	88.3	3.3	1.1	0.0	3.3	1.7	2.2

Figure 4-1: Time per group of tasks



	50-3	50-4	50-5	50-6	100-3	100-4	100-5	100-6	300-3	300-4	300-5	300-6
Ascending	7.8	18.7	13.5	7.7	3.3	72.7	78.8	77.3	1082	1283	2580	3180
Descending	6.7	11.8	4.2	1.7	4.1	65.7	89.9	43.3	1157	1455	1339	2460
DA-setB	11.7	16.5	5.3	6.0	4.2	84.1	89.8	59.2	1297	1162	1371	2280
DA-MaxWind	15.5	31.3	11.2	4.5	9.1	192.2	293.9	156.9	3180	3840	3480	7200
DA-AveWind	15.6	31.4	11.1	4.5	9.1	202.5	342.2	176.3	3120	3840	3480	7200

The Figure 1-1. compares time spent by each algorithm for each group of tasks. The time is increasing exponentially with increase of number of machines or number of tasks (with only exception of the time decrease between 50-3 and 100-3 groups of tasks). Further, Ascending and Descending methods are faster than descending-ascending methods for all groups, with DA-setB method having the running time comparable to Ascending and Descending methods across all groups. However, DA-MaxWind and DA-AveWind methods show at least twice longer time for all groups.

4.7 Conclusion

In this chapter, a discrete optimisation procedure optimisation procedure is presented. This optimisation procedure is an exact method for a class of problems with certain properties. These properties are not too restrictive - we have shown that many NP-hard scheduling problems posses these properties. It also has been shown that the procedure can be potentially applied to problems of multi-objective optimisation. The discrete optimisation procedure is an iterative algorithm, which considers lower bounds on objective function in succession, and on each iteration it either finds a fea-

sible point with this value of objective function, or determines that such a point does not exist. The search for a feasible point can be conducted in several ways which can be viewed as a Descending method, an Ascending or a descending-ascending methods. Application of all three methods to a scheduling problem has been implemented, with three variations of the descending-ascending method. The computational experiments demonstrated that all algorithms are efficient and solve $92\% - 100\%$ of instances to optimality. Further, Descending and Ascending methods perform differently on the same instances, demonstrating that algorithms performance is affected by precedence graph structure. All three descending-ascending methods showed very similar results in terms of number of instances solved to optimality and number of iterations per tasks group, however, DA-setB method showed significantly shorter running time than two other descending-ascending methods.

Chapter 5

The worst-case analysis for an approximation algorithm for a maximum lateness problem

- The results discussed in this chapter have been published in [81]: Julia Memar, Yakov Zinder, and Alexander Kononov. "Worst-Case Analysis of a Modification of the Brucker-Garey-Johnson Algorithm". In International Conference on Optimization Problems and Their Applications, pages 78-92. Springer, 2018.
- The results discussed in this chapter have been presented at the 7th International Conference on Optimization Problems and Their Applications OPTA-2018, Omsk, Russia, July 8-14, 2018.

5.1 Introduction and description of the problem

In this chapter, a polynomial-time approximation algorithm is considered in application to the maximum lateness scheduling problem with parallel identical machines. It is assumed that the tasks are partially ordered, have arbitrary processing times, and the preemptions of tasks' processing are not allowed. The problem is NP-hard in a strong sense, as even a particular case of the problem with all due dates equal

to zero, two machines and the processing times equal to one or two units of time is NP-hard in a strong sense [71].

This polynomial-time approximation algorithm can be viewed as a modification of the Brucker-Garey-Johnson algorithm [7]. A tight worst-case performance guarantee for this algorithm is obtained for the case of the problem when the largest processing time does not exceed the number of machines. The Brucker-Garey-Johnson algorithm was originally developed as an exact algorithm for the unit execution time tasks and precedence constraints in the form of an in-tree. To stress the origin of the presented approximation algorithm, in what follows, it will be referred to as the Brucker-Garey-Johnson algorithm or simply as the BGJ-algorithm. It is also shown that when the largest processing time is greater than the number of machines, the worst-case performance guarantee for the list algorithm, obtained in [46], is tight.

The considered scheduling problem can be stated as follows. A set $N = \{1, \ldots, n\}$ of n tasks is to be processed on $m > 1$ identical machines subject to precedence constraints in the form of an anti-reflexive, anti-symmetric and transitive relation on N. If task i precedes task j in this relation, denoted by $i \to j$, then task i must be completed before task j can be processed. If $i \to j$, then i is called a predecessor of j and j is called a successor of i. The processing of tasks commences at time $t = 0$. Task $i \in N$ requires p_i units of processing time, where p_i is integer. Each task can be processed on any machine. Each machine can process at most one task at a time. If a machine starts processing task i, then it continues to process this task for p_i units of time, i.e. till the completion. Each task $i \in N$ has an associated due date d_i , where d_i is integer. The goal is to minimise the maximum lateness

$$L_{max}(\sigma) = \max_{j \in N}[C_j(\sigma) - d_j], \tag{5.1.1}$$

where $C_j(\sigma)$ is the completion time of task j in schedule σ. In the three-filed notation (see, for example, [6, 87]) the considered problem is denoted by $P|\, prec|\, L_{max}$, where P signifies parallel identical machines, $prec$ indicates the presence of precedence constraints, and L_{max} specifies the objective function, i.e. the criterion of maximum

lateness. If all due dates are zero, the problem is known as a makespan problem and in the three-field notation is denoted by $P|\,prec\,|\,C_{max}$; if all tasks have unit processing times, then the problem is denoted by $P|\,prec, p_j = 1\,|\,L_{max}$; if preemptions are allowed, then the problem is denoted by $P|\,prec, prmp\,|\,L_{max}$. In what follows it is assumed that the largest processing time p_{max} does not exceed the number of machines m.

5.2 BGJ-algorithm

For each task i, the set of all successors of i will be denoted by $K(i)$. That is, $K(i) = \{j \ : i \rightarrow j\}$. Let $d = \max_{i \in N} d_i$. Before constructing a schedule, the BGJ-algorithm computes for each $i \in N$ the axillary priority μ_i as follows:

1. For every task i such that $K(i) = \emptyset$, set $\mu_i = d - d_i$.

2. If all tasks $i \in N$ have been assigned μ_i, then stop. Otherwise, select $i \in N$ such that μ_i has not yet been specified and for each $j \in K(i)$, μ_j has been specified.

3. Set
$$\mu_i = \max\left\{d - d_i, \max_{j \in K(i)} (p_j + \mu_j)\right\} \tag{5.2.1}$$
and go to Step 2.

After obtaining the priorities μ_i for every task i, the BGJ-algorithm constructs a schedule σ, where for each i it uses the priority $\beta_i = p_i + \mu_i$. Let t be the earliest time when a machine is available for a task's processing, and t_k signifies the earliest time, when machine k, $1 \leq k \leq m$, is available for a task's processing. Then, the BGJ-algorithm can be described as follows:

BGJ-algorithm

1. Set $t = t_1 = ... = t_m = 0$.

2. If all tasks have been scheduled, then stop.

3. If no unscheduled task can be assigned for processing at time point t, or there is no machine i such that $t_i = t$, then go to Step 6.

4. Among all unscheduled tasks j, which can be assigned for processing at time point t, choose a task with the largest β_j. Let it be task g. Set $C_g(\sigma) = t + p_g$.

5. Choose any machine i with $t_i = t$ and set $t_i = t + p_g$. Go to Step 2.

6. Set $t = \min_{i \in \{k\,:t_k > t\}} t_i$ and then set $t_i = \max\{t, t_i\}$ for all $1 \leq i \leq m$. Go to Step 3.

Let $S_i(\sigma) = C_i(\sigma) - p_i$ for $i \in N$, i.e. $S_i(\sigma)$ is the starting time of task i in the schedule σ. Following [46], it is convenient to replace the problem of the minimisation of the maximum lateness by the equivalent problem of the minimisation of the criterion $G(\sigma)$ as follows:

$$G(\sigma) = \max_{i \in N}[S_i(\sigma) + \beta_i] = \max_{i \in N}[C_i(\sigma) + \mu_i]. \tag{5.2.2}$$

The following lemma is similar to Lemma 1 in [94] and shows that (5.1.1) can be replaced by (5.2.2).

Lemma 1 *For any schedule σ,*

$$G(\sigma) = L_{max}(\sigma) + d. \tag{5.2.3}$$

Proof: Among all tasks g such that $S_g(\sigma) + \beta_g = G(\sigma)$ select one with the largest $S_g(\sigma)$, say task i. If $\mu_i \neq d - d_i$, then by (5.2.1), there exists $j \in K(i)$ such that $\mu_i = p_j + \mu_j = \beta_j$. Observe that $S_j(\sigma) \geq S_i(\sigma) + p_i$. Hence for this j,

$$G(\sigma) = S_i(\sigma) + \beta_i = S_i(\sigma) + p_i + \mu_i \leq S_j(\sigma) + \mu_i = S_j(\sigma) + \beta_j,$$

which contradicts the choice of i because $S_i(\sigma) < S_j(\sigma)$. Hence, $\mu_i = d - d_i$. On the other hand, (5.2.1) implies that, for any g, $\mu_g \geq d - d_g$. Then,

$$\begin{aligned} L_{max}(\sigma) + d &= \max_{g \in N}[C_g(\sigma) + d - d_g] \leq \max_{g \in N}[C_g(\sigma) + \mu_g] \\ &= G(\sigma) = C_i(\sigma) + \mu_i = C_i(\sigma) + d - d_i \leq L_{max}(\sigma) + d, \end{aligned}$$

which completes the proof. □

5.3 The structure of a schedule

To describe the structure of the schedule σ, constructed by the BGJ-algorithm, it is convenient to introduce the following definitions:

- For any integer t, the slot t is the time interval $[t-1, t]$;
- Let g be the task such that $S_g(\sigma)$ is the smallest starting time among all tasks j such that $S_j(\sigma) + \beta_j = G(\sigma)$;
- A task $i \in N$ is *complete*, if $\beta_i \geq \beta_g$. Otherwise, i is *incomplete*;
- A slot $t \leq S_g(\sigma)$ is *complete* if the number of complete tasks, processed in this slot, equals m;
- A slot $t \leq S_g(\sigma)$, which is not complete, is *incomplete*.
- An incomplete slot t is Type I if at least one of the following holds:

  (t1) in the slot t, at least one machine is idle;
  (t2) there exists an incomplete task j such that $S_j(\sigma) = t - 1$;
  (t3) all tasks j, processed in the slot t, have the same starting times $S_j(\sigma)$.

- An incomplete slot t, which is not Type I, is Type II.

Observe that $S_j(\sigma) \leq S_g(\sigma)$ for any complete task j: by definitions of the task g and a complete task,

$$S_g(\sigma) + \beta_g > S_j(\sigma) + \beta_j, \text{ hence}$$
$$S_g(\sigma) - S_j(\sigma) > \beta_j - \beta_g \geq 0.$$

Lemma 2 *For any Type II slot t, there exists a Type I slot t' such that*

(s1) *$t' < t$;*

(s2) *any slot τ such that $t' < \tau \leq t$ is Type II;*

(s3) *any incomplete task, processed in slot t, is also processed in slot t'.*

Proof: By the definition of a Type II slot, all machines in slot t are busy and at least one machine processes an incomplete task. Among all such incomplete tasks i choose a task with the largest $S_i(\sigma)$. Let it be task j. Then, any incomplete task, processed

in slot t, is also processed in slot $\bar{t} = S_j(\sigma) + 1$. Furthermore, by the definition of a Type I slot, the slot $\bar{t}$ is Type I. Since slot t is Type II, $\bar{t} < t$. The proof is concluded be repeating the procedure for every Type II time slot $\bar{t} < \tau < t$ and choosing t' as the largest integer among all integers $\bar{t}$ such that $\bar{t} < t$ and the slot $\bar{t}$ is Type I. □

For any Type II slot t, the Type I slot t', specified by the conditions (s1)-(s2)-(s3), will be referred to as the *supporting* slot for the slot t.

Lemma 3 *For any Type I slot t and any complete task j such that $S_j(\sigma) \geq t$, there exists a task q such that*

$$C_q(\sigma) \geq t \qquad \textit{and} \qquad q \to j. \tag{5.3.1}$$

Proof: Suppose that either (t1) or (t2) holds or both. Then, the existence of q satisfying (5.3.1) follows from the fact that the BGJ-algorithm has not scheduled j at $t-1$. If (t1) and (t2) do not hold, then there are m tasks processed in the slot t and, by virtue of (t3), all these tasks commence their processing at same point in time $t' < t-1$. Since at least one of these tasks is incomplete and j is not scheduled by the BGJ-algorithm at t', then amongst these m tasks there exists a task q such that $C_q(\sigma) \geq t'+1$ and $q \to j$. Because the same m tasks are processed in slot t' and in slot t, $C_q(\sigma) \geq t$ which gives (5.3.1).□

Corollary 1 *For any Type I slot t and any complete task j such that $S_j(\sigma) \geq t$, there exists a complete task i processed in the slot t and $i \to j$.*

Proof: Among all q, satisfying (5.3.1), select a task with the smallest $S_q(\sigma)$, let it be task i. Since $i \to j$, by virtue of (5.2.1), $\beta_i \geq p_i + \beta_j > \beta_g$, thus i is a complete task. The proof is concluded by the observation that $S_i(\sigma) < t$, because otherwise by Lemma 3 there exists q such that $C_q(\sigma) \geq t$ and $q \to i \to j$, and therefore $q \to j$, which contradicts the selection of i. □

A sequence of tasks j_1, ..., j_k is *a chain* if, for each $1 \leq i < k$, $j_i \to j_{i+1}$. The following corollary is a direct consequence of Corollary 1 and the fact that g is complete.

Corollary 2 *There exists a chain of tasks j_1, ..., j_r such that $j_r \to g$ and, for any Type I incomplete slot, the chain has a task that is processed in this time slot.*

Lemma 4 *At least one complete task is processed in each slot t such that $1 \leq t \leq S_g(\sigma)$.*

Proof: If the slot t is complete, then the lemma follows from the definition of a complete slot. If the slot t is a Type I incomplete slot, the lemma follows from Corollary 1 and the fact that g is a complete task. If the slot t is a Type II incomplete slot, then any incomplete task, processed in slot t, is also processed in its supporting slot (see Lemma 2), which by the definition is a Type I slot. As it has been proven above, at least one complete task is processed in it. Hence, the number of incomplete tasks, processed in the supporting slot and therefore in the slot t, is less than m, and the lemma follows from the definition of a Type II slot which implies that in this slot all m machines are busy. □

Let Z be the set of all Type II slots and $l^z = |Z|$. Denote the set of all supporting Type I slots by Y, and let $l^y = |Y|$. Denote by X the set of Type I slots t which are not in Y, and let $l^x = |X|$. Let c^x, c^y and c^z be the total processing time allocated to complete tasks in the slots of sets X, Y and Z, respectively. Similarly, let e^y and e^z be the total processing time allocated to incomplete tasks in the slots of sets Y and Z, respectively.

Lemma 5 *The following statements hold:*

$$c^z + e^z = ml^z; \tag{5.3.2}$$

$$e^z \leq e^y(m-1). \tag{5.3.3}$$

Proof: By the definition, there is no idle machine in each Type II slot, i.e. the number of tasks processed in the slot is m. Hence the total processing time allocated to complete and incomplete tasks in the l^z Type II slots is ml^z, and (5.3.2) holds. Consider a slot t in Y. The slot is associated with a number of consecutive Type II slots, for which t is a supporting slot. By virtue of Lemma 2, each incomplete task processed in each of these Type II slots is also processed in t. Thus the number of these

Type II slots does not exceed $p_{max}-1$. Furthermore, the number of incomplete tasks processed in each of the Type II slots, is not greater than the number of incomplete tasks processed in t. Thus

$e^z \leq e^y(p_{max}-1) \leq e^y(m-1).\Box$

5.4 Lower Bounds on the Optimal Value of $G(\sigma)$

In what follows, low bounds for the value of $G(\sigma^*)$ are derived, where σ^* is an optimal schedule for $G(\sigma)$.

Let a be the task with the maximum completion time among all complete tasks in σ^*. The total time allocated to incomplete tasks in σ in slots $t \leq S_g(\sigma)$ is at least $e^y + e^z$. Denote by e^* the part of this time, which is allocated in slots $t' \leq C_a(\sigma^*)$ in σ^*. It is easy to see that

$$\max_{j\in N} C_j(\sigma^*) \geq C_a(\sigma^*) + \frac{e^y + e^z - e^*}{m}$$
$$\text{Let } \eta = \frac{e^y + e^z - e^*}{m} \text{ and } \delta = \min\{p_a - p_g, C_a(\sigma^*) - C_g(\sigma^*)\}$$

Since a is complete, $\beta_a \geq \beta_g$, hence $\mu_a \geq \mu_g - (p_a - p_g)$. Thus

$$\begin{aligned} &G(\sigma^*) \geq \max\{C_a(\sigma^*) + \mu_a, C_g(\sigma^*) + \mu_g, C_a(\sigma^*) + \eta\} \geq \\ &\geq C_a(\sigma^*) + \max\{\mu_g - (p_a - p_g), \mu_g - (C_a(\sigma^*) - C_g), \eta\} \geq \\ &\geq C_a(\sigma^*) + \max\{\mu_g - \delta, \eta\}. \end{aligned} \tag{5.4.1}$$

Let l^c be the number of complete slots in σ. By virtue of (5.4.1), Lemmas 4 and 5,

$$\begin{aligned} &G(\sigma^*) \geq C_a(\sigma^*) + \max\{\mu_g - \delta, \eta\} \\ &\geq l^c + \frac{c^x + c^y + c^z + e^* + p_g}{m} + \eta + \max\{\mu_g - \delta - \eta, 0\} \\ &\geq l^c + \frac{l^x + l^y + c^z + e^* + p_g}{m} + \eta + \max\{\mu_g - \delta - \eta, 0\} \\ &= l^c + \frac{l^x + l^y + p_g}{m} + l^z + \frac{e^y}{m} + \max\{\mu_g - \delta - \eta, 0\}. \end{aligned} \tag{5.4.2}$$

Assume that the first slot in σ is incomplete. Then (5.4.2) can be tightened. Let ϱ

be the minimum processing time among all j such that $S_j(\sigma) = 0$. All slots t, such that $1 \le t \le \varrho$, are Type I slots, since they satisfy the condition (t3). By virtue of Corollary 1, $S_j(\sigma^*) \ge \varrho$ for any complete task j with $S_j(\sigma) \ge \varrho$. Denote by c^ϱ the processing time allocated to complete tasks in Type I slots after the point of time ϱ. Then

$$\begin{aligned} C_a(\sigma^*) &\ge \varrho + l^c + \frac{c^\varrho + c^z + e^* + p_g}{m} \ge \varrho + l^c + \frac{l^x + l^y - \varrho + p_g}{m} + \frac{c^z + e^*}{m} \\ &\ge p_{min}\left(1 - \frac{1}{m}\right) + l^c + \frac{l^x + l^y + p_g}{m} + \frac{c^z + e^*}{m}. \end{aligned} \tag{5.4.3}$$

Thus, (5.4.1) and (5.4.3) imply that

$$\begin{aligned} G(\sigma^*) &\ge p_{min}\left(1 - \frac{1}{m}\right) + l^c + \frac{l^x + l^y + p_g}{m} + \frac{c^z + e^*}{m} + \eta + \max\{\mu_g - \delta - \eta, 0\} \\ &= p_{min}\left(1 - \frac{1}{m}\right) + l^c + \frac{l^x + l^y + p_g}{m} + l^z + \frac{e^y}{m} + \max\{\mu_g - \delta - \eta, 0\}. \end{aligned} \tag{5.4.4}$$

Consider a chain $j_1 \to j_2 \to ... \to j_r \to g$, satisfying Corollary 2. Then (5.4.1) can be presented as:

$$\begin{aligned} G(\sigma^*) &\ge C_a(\sigma^*) + \max\{\mu_g - \delta, \eta\} = C_g(\sigma^*) + (C_a(\sigma^*) - C_g(\sigma^*)) + \max\{\mu_g - \delta, \eta\} \\ &\ge \sum_{k=1}^{r} p_{j_k} + p_g + \delta + \max\{\mu_g - \delta, \eta\} \ge l^x + l^y + p_g + \max\{\mu_g, \eta + \delta\}. \end{aligned} \tag{5.4.5}$$

If the first slot in σ is complete, then (5.4.5) can be tightened. Observe that in this case at least p_{min} first time slots in σ are complete. If $S_{j_1}(\sigma) = 0$, then

$$G(\sigma^*) \ge C_g(\sigma^*) + \beta_g \ge \sum_{k=1}^{r} p_{j_k} + \beta_g \ge p_{min} + l^x + l^y + \beta_g.$$

If $S_{j_1}(\sigma) > 0$, then according to the BGJ-algorithm either there exists $h \to j_1$ or there exist another m tasks q with $\beta_q \ge \beta_{j_1}$ and $S_q(\sigma) < S_{j_1}(\sigma)$. Observe that $\beta_{j_1} \ge \sum_{k=1}^{r} p_{j_k} + \beta_g$, thus in these two cases

$$G(\sigma^*) \ge \max_{j \in \{k : \beta_k \ge \beta_{j_1}\}} S_j(\sigma^*) + \beta_{j_1} \ge p_{min} + \beta_{j_1} \ge p_{min} + l^x + l^y + \beta_g.$$

The last two inequalities and (5.4.5) imply that

$$\begin{aligned} G(\sigma^*) &\geq \max\{p_{min} + l^x + l^y + \beta_g, l^x + l^y + p_g + \delta + \eta\} \\ &= l^x + l^y + p_g + \max\{\mu_g + p_{min}, \delta + \eta\}. \end{aligned} \tag{5.4.6}$$

5.5 Worst-Case Performance Guarantee

This lemma will help to simplify the proofs that follow.

Lemma 6 *For nonnegative numbers a, b, h and α, where $\alpha < 1$,*

$$\min\{a, b\} - (1 - \alpha)\max\{a + h, b\} \leq b\alpha - (1 - \alpha)h. \tag{5.5.1}$$

Proof: Consider the following cases: $a \leq b - h$, $b - h < a \leq b$ and $a > b$. In each of these cases (5.5.1) holds:

If $a \leq b - h$ then

$$\min\{a, b\} - (1 - \alpha)\max\{a + h, b\} = a - (1 - \alpha)b \leq b\alpha - h \leq b\alpha - (1 - \alpha)h.$$

If $b - h < a \leq b$, then
$\min\{a, b\} - (1-\alpha)\max\{a+h, b\} = a-(1-\alpha)(a+h) = a\alpha-(1-\alpha)h \leq b\alpha-(1-\alpha)h.$

If $a > b$, then
$\min\{a, b\} - (1-\alpha)\max\{a+h, b\} = b-(1-\alpha)(a+h) < b-(1-\alpha)(b+h) = b\alpha-(1-\alpha)h.$

□
Theorem 1

$$G(\sigma) \leq \left(2 - \frac{1}{m}\right) G(\sigma^*) + \frac{p_{max}}{m} - p_{min}, \tag{5.5.2}$$

and the bound is tight.

Proof: The value of $G(\sigma)$ can be expressed as:

$$G(\sigma) = C_g(\sigma) + \mu_g = l^c + l^x + l^y + l^z + p_g + \mu_g. \tag{5.5.3}$$

If the first slot in σ is complete, then by (5.4.2) and (5.5.3)

$$\begin{aligned} & G(\sigma) - G(\sigma^*) \leq \\ & \leq \left(1 - \frac{1}{m}\right)(l^x + l^y + p_g) + \mu_g - \frac{e^y}{m} - \max\{\mu_g - \delta - \eta, 0\} \\ & = \left(1 - \frac{1}{m}\right)(l^x + l^y + p_g) - \frac{e^y}{m} + \min\{\delta + \eta, \mu_g\}. \end{aligned} \tag{5.5.4}$$

Furthermore, if the first slot in σ is complete, (5.4.6) and (5.5.4) imply that

$$\begin{aligned} & G(\sigma) \leq \\ & \left(2 - \frac{1}{m}\right) G(\sigma^*) - \frac{e^y}{m} + \min\{\delta + \eta, \mu_g\} - \left(1 - \frac{1}{m}\right) \max\{\mu_g + p_{min}, \delta + \eta\} \end{aligned} \tag{5.5.5}$$

If the first slot in σ is incomplete, then by (5.4.4) and (5.5.3)

$$\begin{aligned} & G(\sigma) - G(\sigma^*) \leq \\ & \leq \left(1 - \frac{1}{m}\right)(l^x + l^y + p_g - p_{min}) + \mu_g - \frac{e^y}{m} - \max\{\mu_g - \delta - \eta, 0\} \\ & = \left(1 - \frac{1}{m}\right)(l^x + l^y + p_g - p_{min}) - \frac{e^y}{m} + \min\{\delta + \eta, \mu_g\}. \end{aligned} \tag{5.5.6}$$

It is easy to see that if the first slot in σ is incomplete, (5.4.5) and (5.5.6) imply the same result as (5.5.5) :

$$\begin{aligned} & G(\sigma) \leq \left(2 - \frac{1}{m}\right) G(\sigma^*) - \frac{e^y}{m} + \min\{\delta + \eta, \mu_g\} - \left(1 - \frac{1}{m}\right)(p_{min} + \max\{\mu_g, \delta + \eta\}) \\ & \leq \left(2 - \frac{1}{m}\right) G(\sigma^*) - \frac{e^y}{m} + \min\{\delta + \eta, \mu_g\} - \left(1 - \frac{1}{m}\right) \max\{\mu_g + p_{min}, \delta + \eta\}. \end{aligned}$$

Let $a = \mu_g$, $b = \delta + \eta$, $h = p_{min}$, $\alpha = \frac{1}{m}$. Then by virtue of (5.5.5) and Lemma 6,

$$G(\sigma) \leq \left(2 - \frac{1}{m}\right) G(\sigma^*) - \frac{e^y}{m} + \frac{\delta}{m} + \frac{\eta}{m} - p_{min}\left(1 - \frac{1}{m}\right). \tag{5.5.7}$$

Taking into account (5.3.3),

$$\frac{\eta}{m} - \frac{e^y}{m} = \frac{e^y + e^z - e^*}{m^2} - \frac{e^y}{m} \leq \frac{e^y + e^y(m-1)}{m^2} - \frac{e^y \times m}{m^2} = 0. \tag{5.5.8}$$

Finally, by virtue of (5.5.8) and the fact that $\delta \leq p_{max} - p_{min}$, (5.5.2) is obtained:

$$G(\sigma) \leq \left(2 - \frac{1}{m}\right) G(\sigma^*) + \frac{p_{max} - p_{min}}{m} - p_{min}\left(1 - \frac{1}{m}\right)$$
$$= \left(2 - \frac{1}{m}\right) G(\sigma^*) + \frac{p_{max}}{m} - p_{min}.$$

Observe that by direct substitution of (5.2.3) in (5.5.2), the equivalent bound for the criterion of maximum lateness is obtained:

$$L_{max}(\sigma) \leq \left(2 - \frac{1}{m}\right) L_{max}(\sigma^*) + \left(1 - \frac{1}{m}\right) d + \frac{p_{max}}{m} - p_{min}.$$

To show that (5.5.2) is tight, consider the partially ordered set of tasks depicted by Figure 5-1. The graph constitutes of $km - 1$ identical sections and the last section. On this figure, a p_{min} units task is denoted by p and a p_{max} units task is denoted by p_{max}. Each section constitutes of $m - 1$ rows with $m + 1$ p_{min} units tasks; the m^{th} row of the first $km - 1$ sections is comprised of one p_{min} units task; the m^{th} row of the last section is comprised of one p_{min} units task and m p_{max} units tasks. Hence, the total number of rows is km^2. Priorities of the tasks are the following:

- last $(km^2)^{th}$ row, for the p_{min} units task: $\mu = p_{max} - p_{min}$, $\beta = p_{max}$;
- last $(km^2)^{th}$ row, for the p_{max} units tasks: $\mu = 0$, $\beta = p_{max}$;
- i^{th} row from the bottom, $1 \leq i \leq km^2$, for a p_{min} units task:

$$\mu = p_{max} + (i-2)p_{min}, \quad \beta = p_{max} + (i-1)p_{min}.$$

The schedule σ constructed by the BGJ-algorithm and the optimal schedule σ^* are depicted by Figure 5-2. To demonstrate that the σ^* is optimal, it is shown below that $G(\sigma^*)$ is equal to a lower bound for its value. Let l be the l^{th} row from the top of a section h, where $1 \leq l \leq m$ and $1 \leq h \leq km$, and j be the j^{th} row from the top,

$1 \leq j \leq km^2$. Then the following equalities hold:

$$\begin{aligned} j &= km^2 - i + 1; \\ j &= l + (h-1)m, \text{ hence} \\ i &= km^2 + 1 - l - (h-1)m \end{aligned} \tag{5.5.9}$$

Hence, β for p_{min} tasks in l^{th} row from the top of a section h, where $1 \leq l \leq m$ and $1 \leq h \leq km$, is calculated as

$$\begin{aligned} &\beta = p_{max} + (i-1)p_{min} = p_{max} + \left(km^2 + 1 - l - (h-1)m - 1\right) p_{min} \\ &= p_{max} + (km^2 - l - (h-1)m)p_{min} \end{aligned}$$

Value of $G(\sigma)$:

To determine the value of $G(\sigma)$, consider the following cases:

- $l = m$, $h = km$, p_{min} units task;
- $l = m$, $h = km$, p_{max} units tasks;
- $1 \leq l \leq m-1$, $1 \leq h \leq km$, a "shaded" p_{min} units task;
- $l = m$, $1 \leq h \leq km - 1$, p_{min} units task;

Consider the p_{min} units task in the last row of the last section, denote the task by g:

$$\begin{aligned} &S_g(\sigma) + \beta_g = \underbrace{[2p_{min}(m-1) + p_{min}]}_{per\ section} \underbrace{(km-1)}_{km-1 sections} \\ &+ \underbrace{2p_{min}(m-1) + p_{max}}_{last\ section} + \underbrace{p_{max}}_{\beta_g} \\ &= 2p_{min}km^2 - p_{min}km + 2p_{max} - p_{min} \end{aligned} \tag{5.5.10}$$

Consider a p_{max} units task i in the last row of the last section:

$$\begin{aligned} &S_i(\sigma) + \beta_i = \underbrace{[2p_{min}(m-1) + p_{min}]}_{per\ section} \underbrace{(km-1)}_{km-1 sections} + \\ &+ \underbrace{2p_{min}(m-1)}_{last\ section} + \underbrace{p_{max}}_{\beta_i} \\ &= 2p_{min}km^2 - p_{min}km + p_{max} - p_{min} \end{aligned} \tag{5.5.11}$$

Let i be the "shaded" task with the largest $S_i(\sigma)$ out of all tasks in the row l of a section h, $1 \le l \le m-1$, $1 \le h \le km$:

$$\begin{aligned}
&S_i(\sigma) + \beta_i = \\
&= \underbrace{2p_{min}(l-1) + p_{min} + (h-1)\left(2p_{min}(m-1) + p_{min}\right)}_{S_i(\sigma)} + \underbrace{p_{max} + (km^2 - l - (h-1)m)p_{min}}_{\beta_i} \\
&= p_{max} + p_{min}l + hmp_{min} - hp_{min} - mp_{min} + p_{min}km^2 \\
&= p_{max} + hp_{min}(m-1) + p_{min}(l-m) + p_{min}km^2 \\
&\le p_{max} + kmp_{min}(m-1) + p_{min}(m-1-m) + p_{min}km^2 \\
&= 2p_{min}km^2 - p_{min}km + p_{max} - p_{min}. \qquad (5.5.12)
\end{aligned}$$

Let i be the task in the row m of a section h, $1 \le h \le km-1$:

$$\begin{aligned}
&S_i(\sigma) + \beta_i = \\
&= \underbrace{2p_{min}(m-1) + (h-1)\left(2p_{min}(m-1) + p_{min}\right)}_{S_i(\sigma)} + \underbrace{p_{max} + (km^2 - m - (h-1)m)p_{min}}_{\beta_i} \\
&= p_{max} + p_{min}km^2 - p_{min} + hmp_{min} - hp_{min} \\
&\le 2p_{min}km^2 - p_{min}km + p_{max} - mp_{min} < 2p_{min}km^2 - p_{min}km + p_{max} - p_{min} \qquad (5.5.13)
\end{aligned}$$

By virtue of (5.5.10)-(5.5.13),

$$G(\sigma) = 2p_{min}km^2 - p_{min}km + 2p_{max} - p_{min} \qquad (5.5.14)$$

Value of $G(\sigma^*)$:

To determine the value of $G(\sigma^*)$, consider the following cases:

- $l = m$, $h = km$, p_{max} units tasks;
- $1 \le l \le m-1$, $1 \le h \le km$, p_{min} units tasks;
- $l = m$, $1 \le h \le km$, p_{min} units tasks;

Consider a p_{max} units task a in the last row of the last section:

$$\begin{aligned}
&S_a(\sigma^*) + \beta_a = \\
&= \underbrace{mp_{min}}_{\text{per section}} \underbrace{(km-1)}_{km-1 \text{ sections}} + \underbrace{mp_{min}}_{\text{last section}} + \underbrace{p_{max}}_{\beta_a} \\
&= kp_{min}m^2 + p_{max} \qquad (5.5.15)
\end{aligned}$$

Let i be the task with the largest $S_i(\sigma)$ out of all tasks in the row l of a section h, $1 \leq l \leq m-1$, $1 \leq h \leq km$:

$$\begin{aligned} & S_i(\sigma^*) + \beta_i = \\ & = \underbrace{(h-1)mp_{min} + lp_{min}}_{S_i(\sigma^*)} + \underbrace{p_{max} + (km^2 - l - (h-1)m)p_{min}}_{\beta_i} \\ & = kp_{min}m^2 + p_{max} \end{aligned} \tag{5.5.16}$$

Let i be a p_{min} units task in the row m of a section h, $1 \leq h \leq km$:

$$\begin{aligned} & S_i(\sigma^*) + \beta_i = \\ & = \underbrace{(h-1)mp_{min} + (m-1)p_{min}}_{S_i(\sigma^*)} + \underbrace{p_{max} + (km^2 - m - (h-1)m)p_{min}}_{\beta_i} \\ & = kp_{min}m^2 + p_{max} - p_{min} \end{aligned} \tag{5.5.17}$$

Observe that for the schedule σ^* the following lower bound holds:

$$\begin{aligned} G(\sigma^*) & \geq \frac{\sum_{i \in N} p_i}{m} \\ & = \frac{\underbrace{km}_{sections} \times \underbrace{((m-1)(m+1)+1)}_{per\ section} \times p_{min} + mp_{max}}{m} \\ & = kp_{min}m^2 + p_{max} \end{aligned} \tag{5.5.18}$$

Finally, by virtue of (5.5.15)-(5.5.17),

$$G(\sigma^*) = kp_{min}m^2 + p_{max}, \tag{5.5.19}$$

and by virtue of (5.5.18), σ^* is the optimal schedule.
To complete the proof, we substitute (5.5.14) and (5.5.19) in (5.5.2) and show that

the bound is tight:

$$\left(2-\frac{1}{m}\right)G(\sigma^*)+\frac{p_{max}}{m}-p_{min}=\left(2-\frac{1}{m}\right)(kp_{min}m^2+p_{max})+\frac{p_{max}}{m}-p_{min}$$

$$=2kp_{min}m^2+2p_{max}-kp_{min}m-\frac{p_{max}}{m}+\frac{p_{max}}{m}-p_{min}=G(\sigma).$$

□

5.6 The case of the problem when $p_{max} > m$

To establish that (5.5.2) does not hold when $p_{max} > m$ consider the partially ordered set of tasks depicted by Figure 5-3 and let $p_{max} = m+1$ and $p_{min} = 1$. The graph constitutes of m identical sections and the last section. Each of the first m sections constitutes of $m-1$ rows with $m+1$ unit tasks and the m^{th} row, which is comprised of one unit task and $m-1$ $(m+1)$ units tasks. The last section constitutes of $m-1$ rows with $m+1$ unit tasks and the m^{th} row, which is comprised of one unit task and m $(m+1)$ units tasks. Priorities of the tasks are the following:

- last $(m^2+m)^{th}$ row: for the unit task $\mu = m^2+m-1$, $\beta = m^2+m$;
- last $(m^2+m)^{th}$ row: for the $(m+1)$ units tasks $\mu = m^2-1$, $\beta = m^2+m$;
- i^{th} row from the bottom, $1 \le i \le m^2+m$, for a unit task in: $\mu = m^2+m+i-2$, $\beta = m^2+m+i-1$;
- the m^{th} row in the first m sections, for a $(m+1)$ units task: $\mu = 0$, $\beta = m+1$.

The schedule σ constructed by the BGJ-algorithm and the optimal schedule σ^* are depicted by Figure 5-4. Similar to the previous subsection, let l be the l^{th} row from the top of a section h, where $1 \le l \le m$ and $1 \le h \le m+1$, and j be the j^{th} row from the top, $1 \le j \le m^2+m$. Then the following equalities hold:

$$\begin{aligned} &j = m^2+m-i+1;\\ &j = l+(h-1)m, \text{ hence}\\ &i = m^2+m+1-l-(h-1)m = m^2+2m+1-l-hm. \end{aligned} \tag{5.6.1}$$

Figure 5-1: Set of tasks: $p_{max} \leq m$

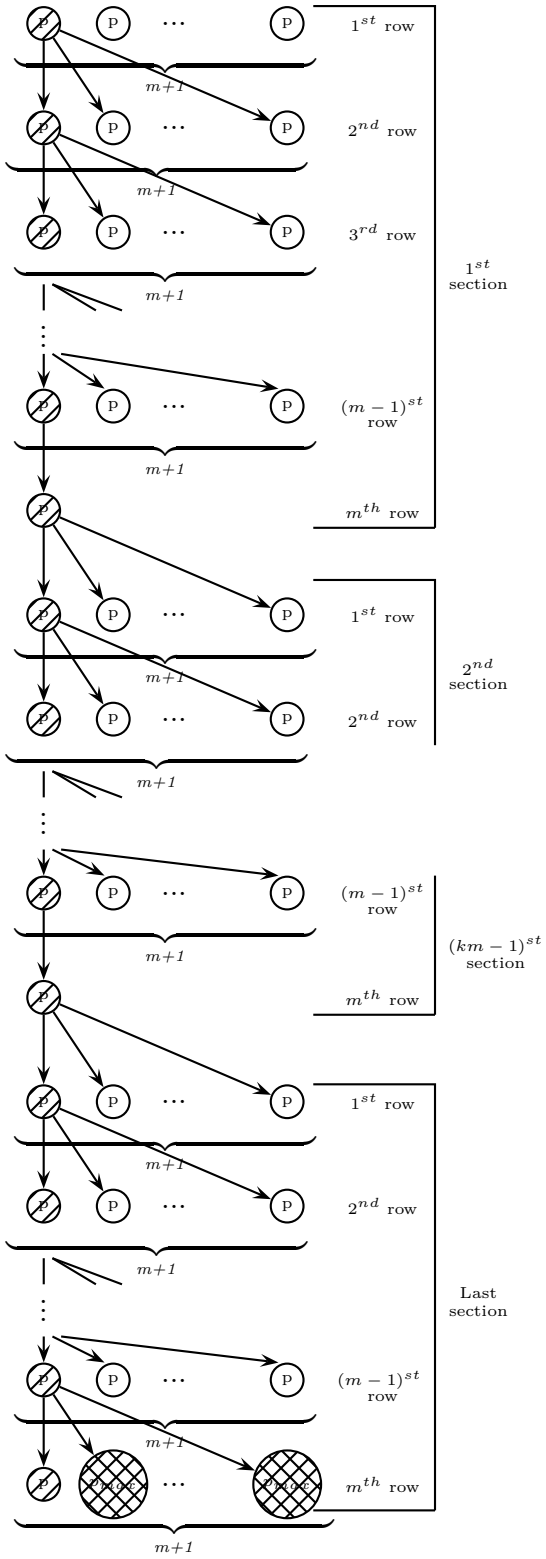
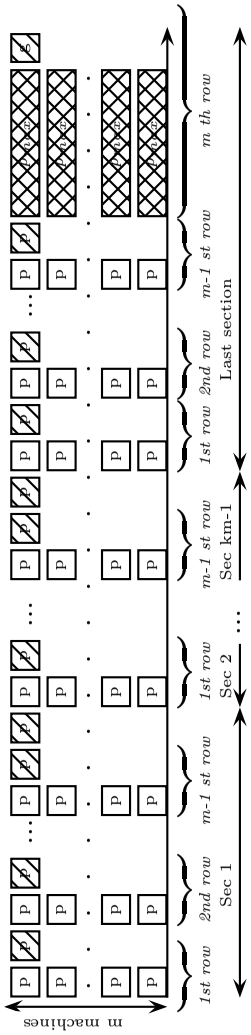
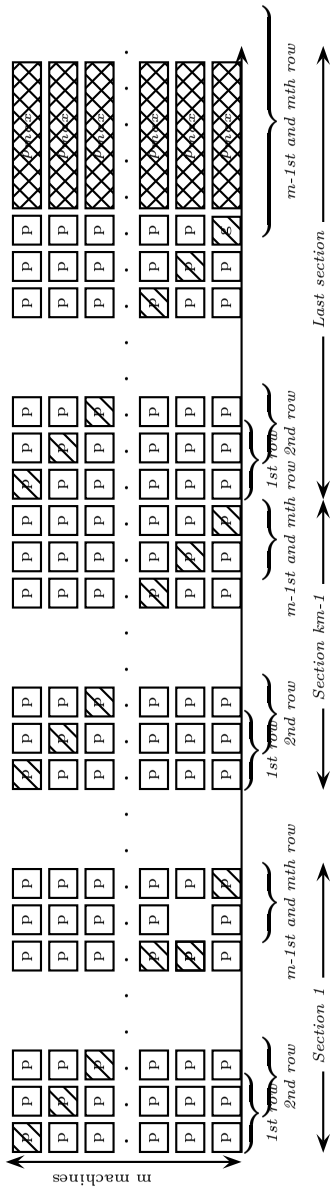


Figure 5-2: Schedules: $p_{max} \leq m$



Schedule σ,
constructed by the BGJ-algorithm



Optimal schedule σ^*

Hence, β for unit tasks in l^{th} row from the top of a section h, where $1 \leq l \leq m$ and $1 \leq h \leq m+1$, is calculated as

$$\beta = m^2 + m + i - 1 = m^2 + m - 1 + m^2 + 2m + 1 - l - hm = 2m^2 + 3m - l - hm.$$

Value of $G(\sigma)$:

To determine the value of $G(\sigma)$, consider the following cases:

- $h = 1$, $1 \leq l \leq m-1$, a "shaded" unit task;
- $2 \leq h \leq m+1$, $l = 1$, a "shaded" unit task;
- $2 \leq h \leq m+1$, $2 \leq l \leq m-1$ a "shaded" unit task;
- $2 \leq h \leq m$, $l = m$, a "shaded" unit task;
- $2 \leq h \leq m$, $l = m$, an $(m+1)$ units task;
- $h = m+1$, $l = m$, an $(m+1)$ units task;
- $h = m+1$, $l = m$, the unit task.

Let i be the "shaded" task with the largest $S_i(\sigma)$ out of all tasks in the l^{th} row of the first section, $1 \leq l \leq m-1$:

$$\begin{aligned} S_i(\sigma) + \beta_i &= \underbrace{2l-1}_{S_i(\sigma)} + \underbrace{2m^2 + 3m - l - m}_{\beta_i} \\ &= l + 2m^2 + 2m - 1 < 2m^2 + 3m - 2. \end{aligned} \tag{5.6.2}$$

Let i be the "shaded" unit task with the largest $S_i(\sigma)$ out of all tasks in the first row of the section h, $2 \leq h \leq m+1$:

$$\begin{aligned} &S_i(\sigma) + \beta_i = \\ &= \underbrace{2m - 1 + (h-2)(m+1+2(m-2)+1) + m}_{S_i(\sigma)} + \underbrace{2m^2 + 3m - 1 - hm}_{\beta_i} \\ &= 2m^2 + 2h(m-1) + 2 \\ &\leq 2m^2 + 2(m^2 - 1) + 2 = 4m^2. \end{aligned} \tag{5.6.3}$$

Let i be the "shaded" unit task with the largest $S_i(\sigma)$ out of all tasks in the l^{th} row of the section h, $2 \leq l \leq m-1$, $2 \leq h \leq m+1$:

$$\begin{aligned}
&S_i(\sigma) + \beta_i = \\
&= \underbrace{2m - 1 + (h-2)(m+1+2(m-2)+1) + m + 1 + 2l - 1}_{S_i(\sigma)} + \underbrace{2m^2 + 3m - l - hm}_{\beta_i} \\
&= 2m^2 + 2h(m-1) + l + 3 \\
&\leq 2m^2 + 2(m^2-1) + m - 1 + 3 = 4m^2 + m. \qquad (5.6.4)
\end{aligned}$$

Let i be the unit task in the m^{th} row of the section h, $1 \leq h \leq m$:

$$\begin{aligned}
&S_i(\sigma) + \beta_i = \\
&= \underbrace{2m - 1 + (h-2)(m+1+2(m-2)+1) + m + 1 + 2(m-2)}_{S_i(\sigma)} + \underbrace{2m^2 + 3m - m - hm}_{\beta_i} \\
&= 2m^2 + 2h(m-1) + m \\
&\leq 2m^2 + 2(m^2-1) + m = 4m^2 + m - 2. \qquad (5.6.5)
\end{aligned}$$

Let i be an $(m+1)$ units task in the m^{th} row of the section h, $1 \leq h \leq m$:

$$\begin{aligned}
&S_i(\sigma) + \beta_i = \\
&= \underbrace{2m - 1 + (h-2)(m+1+2(m-2)+1) + m + 1 + 2(m-2)}_{S_i(\sigma)} + \underbrace{m+1}_{\beta_i} \\
&= 1 + h(3m-2) \leq 1 + m(3m-2) = 3m^2 - 2m + 1. \qquad (5.6.6)
\end{aligned}$$

Let i be an $(m+1)$ units task in the m^{th} row of the last section:

$$\begin{aligned}
&S_i(\sigma) + \beta_i = \\
&= \underbrace{2m - 1 + (m-1)(m+1+2(m-2)+1) + m + 1 + 2(m-2)}_{S_i(\sigma)} + \underbrace{m^2 + m}_{\beta_i} \\
&= 4m^2 + m - 2. \qquad (5.6.7)
\end{aligned}$$

Let g be the unit task in the m^{th} row of the last section:

$$\begin{aligned}
&S_g(\sigma) + \beta_g = \\
&= \underbrace{2(m-1)+1}_{1^{st}\ section} + \underbrace{[m+1+2(m-2)+1](m-1)}_{2^{nd}-m^{th}\ sections} + \\
&\underbrace{m+1+2(m-2)+m+1}_{the\ last\ section} + \underbrace{m^2+m}_{\beta_g} = 4m^2+2m-1
\end{aligned} \tag{5.6.8}$$

By virtue of (5.6.2)-(5.6.8),

$$G(\sigma) = 4m^2 + 2m - 1 \tag{5.6.9}$$

Value of $G(\sigma^*)$:

To determine the value of $G(\sigma^*)$, consider the following cases:

- $1 \le l \le m-1$, $1 \le h \le m+1$, a unit task;
- $l = m$, $1 \le h \le m+1$, a unit task;
- $l = m$, $1 \le h \le m$, an $(m+1)$ units task;
- $l = m$, $h = m+1$, an $(m+1)$ units task.

Let i be the unit task with the largest $S_i(\sigma)$ out of all tasks in the l^{th} row of the section h, $1 \le l \le m-1$, $1 \le h \le m+1$:

$$\begin{aligned}
&S_i(\sigma^*) + \beta_i = \\
&= \underbrace{m(h-1)+l}_{S_i(\sigma^*)} + \underbrace{2m^2+3m-l-hm}_{\beta_i} \\
&= 2m^2 + 2m.
\end{aligned} \tag{5.6.10}$$

Let i be a unit task in the m^{th} row of the section h, $1 \leq h \leq m+1$:

$$\begin{aligned} & S_i(\sigma^*) + \beta_i = \\ & = \underbrace{m(h-1) + m - 1}_{S_i(\sigma^*)} + \underbrace{2m^2 + 3m - m - hm}_{\beta_i} \\ & = 2m^2 + 2m - 1. \end{aligned} \tag{5.6.11}$$

Let a be the $(m+1)$ units task with the largest starting time out of all such tasks from sections $1-m$. Recall that $\mu_a = 0$ and that there are $m \times (m-1)$ such tasks:

$$\begin{aligned} & C_a(\sigma^*) + \mu_a = \\ & = \underbrace{m}_{per\ section} \times \underbrace{(m+1)}_{m+1\ sections} + \underbrace{m+1}_{long\ tasks\ of\ last\ section} + \underbrace{(m-1) \times (m+1)}_{(m-1)\ \ columns\ of\ (m+1)\ unit\ tasks} \\ & = 2m^2 + 2m \end{aligned} \tag{5.6.12}$$

Let i be an $(m+1)$ units task from the last section:

$$\begin{aligned} & C_i(\sigma^*) + \mu_i = \\ & = \underbrace{m}_{per\ section} \times \underbrace{(m+1)}_{m+1\ sections} + \underbrace{m+1}_{long\ tasks\ of\ last\ section} + \underbrace{m^2 - 1}_{\mu_i} \\ & = 2m^2 + 2m \end{aligned} \tag{5.6.13}$$

By virtue of (5.6.10)-(5.6.13),

$$G(\sigma^*) = 2m^2 + 2m \tag{5.6.14}$$

Observe that similar to (5.5.18), this value is equal to the lower bound on the value

of $G(\sigma^*)$, and hence the σ^* is optimal:

$$G(\sigma^*) \geq \frac{\sum_{i \in N} p_i}{m}$$

$$= \frac{\underbrace{(m+1)}_{sections} \times \underbrace{((m-1)(m+1)+1)}_{per\ section} + \underbrace{m}_{1^{st}-m^{th}\ sections} \times \underbrace{((m-1)(m+1))}_{long\ tasks,\ per section} + \underbrace{m}_{last\ section} \times \underbrace{(m+1)}_{long\ tasks}}{m}$$

$$= 2m^2 + 2m$$

When (5.6.9) and (5.6.14) are substituted in (5.5.2), the bound does not hold:

$$\left(2 - \frac{1}{m}\right) G(\sigma^*) + \frac{p_{max}}{m} - p_{min} = \left(2 - \frac{1}{m}\right)(2m^2 + 2m) + \frac{m+1}{m} - 1$$

$$= 4m^2 + 4m - 2m - 2 + 1 + \frac{1}{m} - 1 = G(\sigma) - (1 - \frac{1}{m}) < G(\sigma).$$

This section is concluded by demonstrating that if $p_{max} > m$, then there exists a sequence of instances of the considered maximum lateness problem such that, for the corresponding sequence of optimal schedules $\sigma_1^*, \sigma_2^*, ..., \sigma_k^*, ...$ and the sequence of schedules $\sigma_1, \sigma_2, ..., \sigma_k, ...$, constructed by the BGJ-algorithm

$$G(\sigma_k) \to 2G(\sigma_k^*).$$

Consider the set of tasks constituting of m p_{max} units tasks and one p_{min} units task. For any task j such that $p_j = p_{max}$ let $\mu_j = 0$, then $\beta_j = p_{max}$. Denote p_{min} units task by g and let $\mu_g = p_{max} - p_{min}$, then $\beta_g = p_{max}$. Schedule σ constructed by the BGJ-algorithm and an optimal schedule are depicted on the figure 5-5. The BGJ-algorithm could assign $S_j(\sigma) = 0$ for any task j such that $p_j = p_{max}$ and $S_g(\sigma) = p_{max}$. Then

$$G(\sigma) = C_g(\sigma) + \mu_g = \underbrace{p_{max} + p_{min}}_{C_g(\sigma)} + \underbrace{p_{max} - p_{min}}_{\mu_g} = 2p_{max}$$

In the optimal schedule σ^* $S_g(\sigma^*) = 0$, and $S_j(\sigma^*) = 0$ for $m - 1$ tasks j such that

Figure 5-3: Set of tasks: $p_{max} > m$

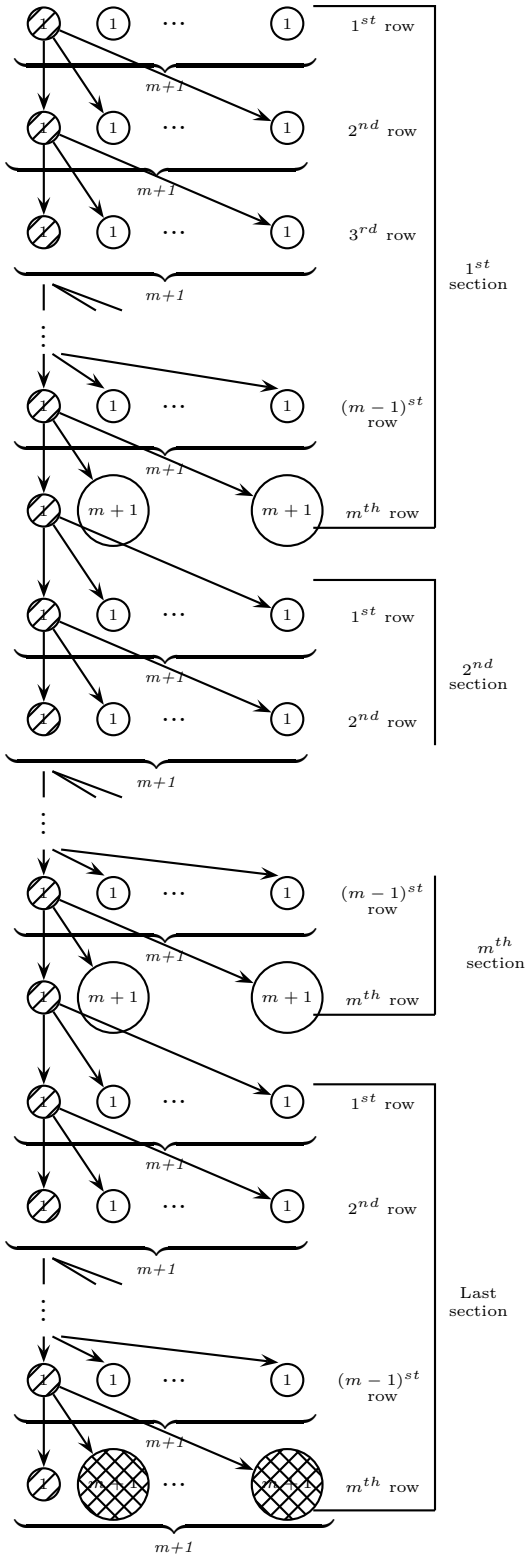
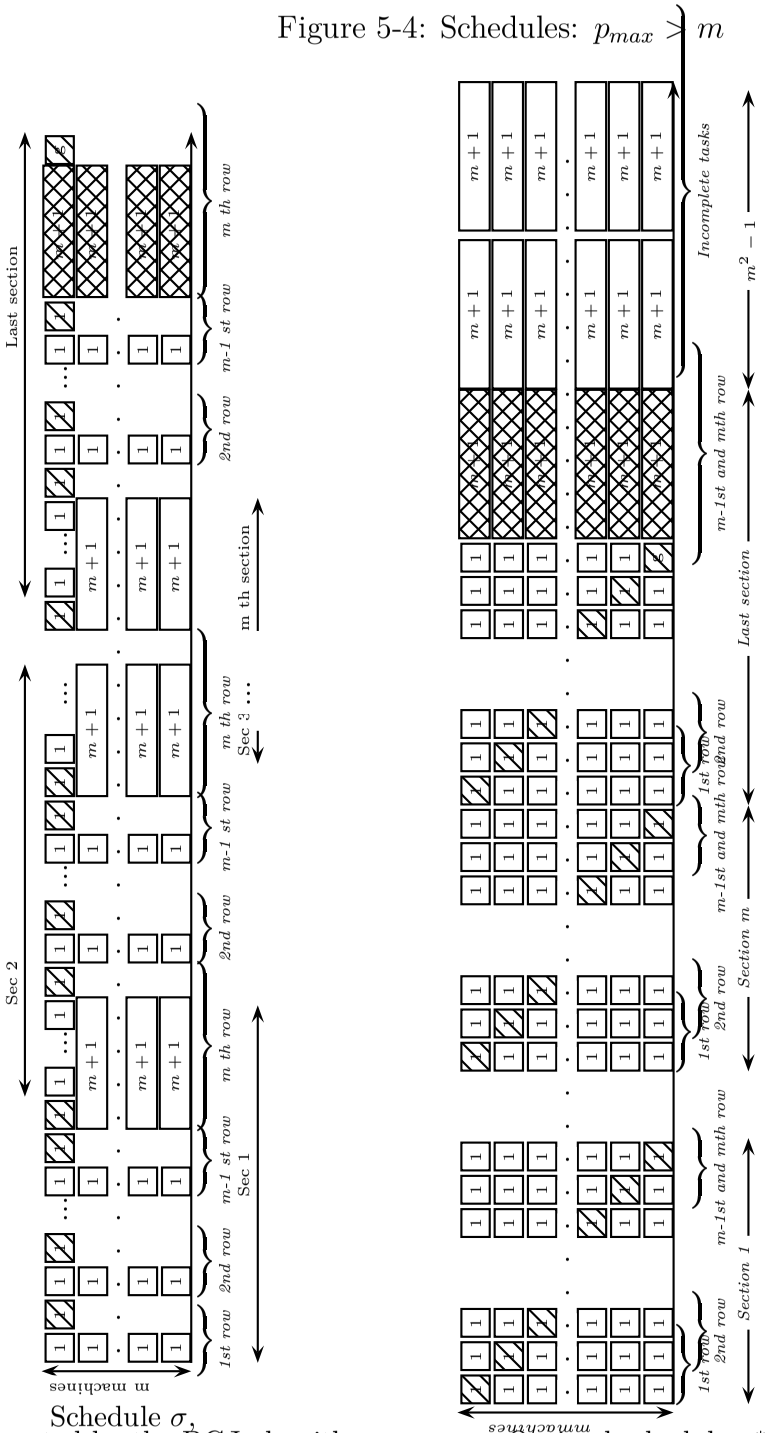


Figure 5-4: Schedules: $p_{max} \geq m$



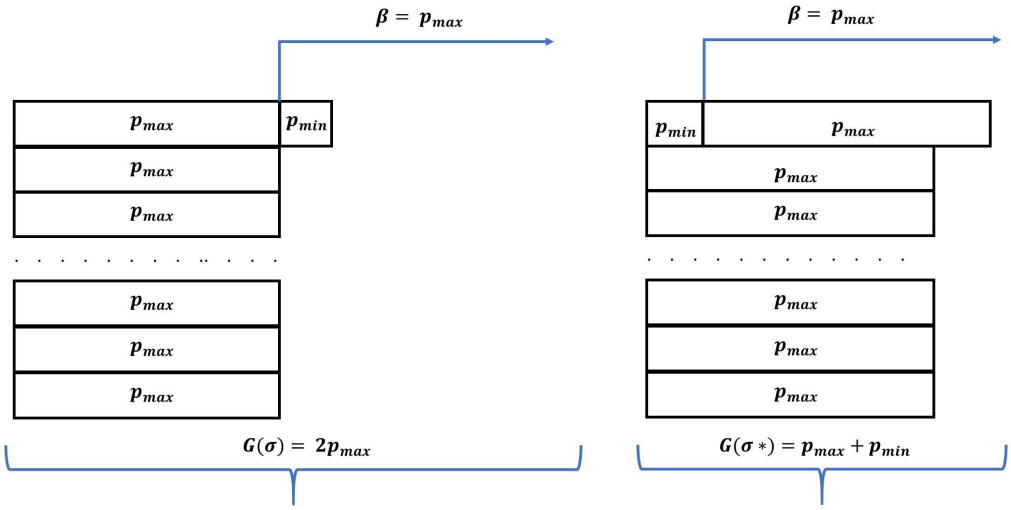
Schedule σ, constructed by the BGJ-algorithm

Optimal schedule σ^*

$p_j = p_{max}$. For one task i such that $p_i = p_{max}$ $S_i(\sigma^*) = p_{min}$. Then

$$G(\sigma^*) = C_a(\sigma^*) + \mu_a = p_{min} + p_{max};$$
$$G(\sigma) = \frac{2p_{max}}{p_{max} + p_{min}} G(\sigma^*) = \left(2 - \frac{2p_{min}}{p_{max} + p_{min}}\right) G(\sigma^*) \to 2G(\sigma^*).$$

Figure 5-5: $G(\sigma) \to 2G(\sigma^*)$



5.7 Conclusion

In this chapter, a polynomial-time approximation algorithm is presented for a maximum lateness problem with parallel identical machines with precedence constraints and tasks with arbitrary processing times. This algorithm can be viewed as a modification of the Brucker-Garey-Johnson algorithm. For the case of the problem when the maximum processing time does not exceed the number of parallel machines, a tight worst-case performance guarantee is derived. It is shown that if the maximum processing time exceeds the number of machines, this guarantee does not hold. Furthermore, in this case the algorithm is a 2-approximation algorithm, and the bound is asymptotically tight.

Chapter 6

Conclusion and further research

In this thesis several algorithms for NP-hard scheduling problems are introduced and discussed. This thesis contributes to the existing research as follows:

- A new approach that utilises Lagrangian relaxation and decomposition techniques is developed for flow shop problems with a job-dependent buffer - these techniques have never been used before for such problems. Efficient Lagrangian relaxation and decomposition-based heuristics are designed; the efficiency of these heuristics tested computationally. The considered problems belong to a class of flow shop problems with a job-dependent buffer - the new area of research that has gained attention only in the last 10-15 years. In these problems, the storage requirement varies from job to job and a job occupies the storage continuously from the start of its first operation till the completion of its second operation rather than only between operations.

- A new discrete optimisation procedure is introduced. This optimisation procedure can be viewed as an alternative to the widely used exact algorithm - the branch and bound algorithm, for a class of discrete optimisation problems with certain properties. These properties are not too restrictive as they allow to include in this class several NP-hard scheduling problems. The procedure is generalisation of the method proposed in [112, 113], where it has been shown that the method outperforms a conventional branch and bound algorithm for the considered scheduling problems. This discrete optimisation procedure is an

iterative algorithm, that searches for a feasible solution with the objective value of the current lower bound or determines that such solution does not exists. The various methods of how this search can be carried out are investigated , and these methods are compared computationally in application to a scheduling problem.

- A worst-case analysis of an approximation algorithm for a classical maximum lateness problem on parallel machines is presented. Though there is a wealth of literature for the case of the problem when all tasks are of unit execution time, much less is know about more general case of the problem - when there are arbitrary processing times and arbitrary precedence constraints. This thesis addressed this gap in the literature by presenting a polynomial-time approximation algorithm that can be viewed as a modification of the Brucker-Garey-Johnson algorithm [7]. The Brucker-Garey-Johnson algorithm was originally developed as an exact algorithm for the case of the problem with unit execution time tasks and precedence constraints represented by an in-tree. A tight a worst-case performance guarantee is obtained for the case of the problem, when the largest processing time does not exceed the number of machines. The guarantee is tight for arbitrary large instances of the considered maximum lateness problem. It is shown that, if the largest processing time is greater than the number of machines, then the classical worst-case performance guarantee for the list algorithm, obtained in [46], is tight.

Further research will focus on the following:

- attempt developing approximation algorithms and analyse their worst-case performance for flow shop problems with the job-dependent buffer considered in this thesis;

- investigate the possibility of obtaining a worst-case performance guarantee for other priority algorithms such as the Garey-Johnson algorithm and the Zinder-Roper algorithm for the case of arbitrary processing times;

- consider an application of the discrete optimisation procedure to other scheduling problems, such as problems on parallel machines with arbitrary processing times.

Bibliography

[1] Meral Azizoglu and Omer Kirca. Tardiness minimization on parallel machines. *International Journal of Production Economics*, 55(2):163–168, 1998.

[2] Philippe Baptiste, Antoine Jouglet, and David Savourey. Lower bounds for parallel machine scheduling problems. *International Journal of Operational Research*, 3(6):643–664, 2008.

[3] Rabah Belaid, Vincent T'kindt, and Carl Esswein. Scheduling batches in flow-shop with limited buffers in the shampoo industry. *European Journal of Operational Research*, 223(2):560–572, 2012.

[4] Joanna Berlińska, Alexander Kononov, and Yakov Zinder. Two-machine flow shop with dynamic storage space. *Optimisation Letters*, pages 1–22, 2020.

[5] Bertrand Braschi and Denis Trystram. A new insight into the coffman–graham algorithm. *SIAM Journal on Computing*, 23(3):662–669, 1994.

[6] Peter Brucker. Scheduling algorithms 5th edition, 2006.

[7] Peter Brucker, Michael R. Garey, and David S. Johnson. Scheduling equal-length tasks under treelike precedence constraints to minimize maximum lateness. *Mathematics of Operations Research*, 2(3):275–284, 1977.

[8] Peter Brucker, Silvia Heitmann, and Johann Hurink. Flow-shop problems with intermediate buffers. *OR Spectrum*, 25(4):549–574, 2003.

[9] Peter Brucker and Sigrid Knust. *Complex Scheduling*. Springer, 2012.

[10] Herbert G. Campbell, Richard A. Dudek, and Milton L. Smith. A heuristic algorithm for the n job, m machine sequencing problem. *Management science*, 16(10):B–630, 1970.

[11] Jacques Carlier. Scheduling jobs with release dates and tails on identical machines to minimize the makespan. *European Journal of Operational Research*, 29(3):298–306, 1987.

[12] Edward G. Coffman and Ronald L. Graham. Optimal scheduling for two-processor systems. *Acta informatica*, 1(3):200–213, 1972.

[13] Edward G. Coffman, Jr, Michael R. Garey, and David S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7(1):1–17, 1978.

[14] Richard W. Conway, William L. Maxwell, and Louis W. Miller. *Theory of scheduling*. Courier Corporation, 2003.

[15] Stephen Cook. The P versus NP problem. *The millennium prize problems*, pages 87–104, 2006.

[16] Ding-Zhu Du and Panos M. Pardalos. *Handbook of combinatorial optimisation: supplement*, volume 1. Springer Science & Business Media, 2013.

[17] Simon Dunstall and Andrew Wirth. A comparison of branch-and-bound algorithms for a family scheduling problem with identical parallel machines. *European Journal of Operational Research*, 167(2):283–296, 2005.

[18] Emrah B. Edis and Ceyda Oguz. Parallel machine scheduling with additional resources: a lagrangian-based constraint programming approach. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimisation Problems*, pages 92–98. Springer, 2011.

[19] Hamilton Emmons and George Vairaktarakis. *Flow Shop Scheduling*. Springer, 2013.

[20] Andreas Ernst, Joey Fung, Gaurav Singh, and Yakov Zinder. Flexible flow shop with dedicated buffers. *Discrete Applied Mathematics*, 2018.

[21] Marshall L. Fisher. An applications oriented guide to Lagrangian relaxation. *Interfaces*, 15(2):10–21, 1985.

[22] Marshall L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management science*, 50(12_supplement):1861–1871, 2004.

[23] Jose M. Framinan, Jatinder ND. Gupta, and Rainer Leisten. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society*, 55(12):1243–1255, 2004.

[24] Joey Fung, Gaurav Singh, and Yakov Zinder. Capacity planning in supply chains of mineral resources. *Information Sciences*, 316:397–418, 2015.

[25] Joey Fung and Yakov Zinder. Permutation schedules for a two-machine flow shop with storage. *Operations Research Letters*, 44(2):153–157, 2016.

[26] Devdatta Gangal and Abhiram Ranade. Precedence constrained scheduling in $(2-7/(3p+1)$ optimal. *Journal of Computer and System Sciences*, 74(7):1139–1146, 2008.

[27] Michael R. Garey and David S. Johnson. Scheduling tasks with nonuniform deadlines on two processors. *Journal of the ACM (JACM)*, 23(3):461–467, 1976.

[28] Michael R. Garey and David S. Johnson. Two-processor scheduling with start-times and deadlines. *SIAM journal on Computing*, 6(3):416–426, 1977.

[29] Michael R. Garey and David S. Johnson. "Strong"NP-Completeness Results: Motivation, Examples, and Implications. *Journal of the ACM (JACM)*, 25(3):499–508, 1978.

[30] Michael R. Garey and David S. Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[31] Michael R. Garey, David S. Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2):117–129, 1976.

[32] Arthur M. Geoffrion. Lagrangian relaxation and its uses in integer programming. *Western Management Science Institute, University of California at Los Angeles Working Paper*, (195).

[33] Arthur M. Geoffrion. Lagrangean relaxation for integer programming. In *Approaches to integer programming*, pages 82–114. Springer, 1974.

[34] Paul C. Gilmore and Ralph E. Gomory. Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations research*, 12(5):655–679, 1964.

[35] Teofilo Gonzalez and Sartaj Sahni. Flowshop and jobshop schedules: complexity and approximation. *Operations research*, 26(1):36–52, 1978.

[36] Ronald L. Graham. Bounds for certain multiprocessing anomalies. *Bell system technical journal*, 45(9):1563–1581, 1966.

[37] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

[38] Hanyu Gu, Alexander Kononov, Julia Memar, and Yakov Zinder. Efficient lagrangian heuristics for the two-stage flow shop with job dependent buffer requirements. *Journal of Discrete Algorithms*, 52-53:143 – 155, 2018.

[39] Hanyu Gu, Julia Memar, and Yakov Zinder. Scheduling batch processing in flexible flowshop with job dependent buffer requirements: Lagrangian relaxation approach. In *International Workshop on Algorithms and Computation*, pages 119–131. Springer, 2018.

[40] Hanyu Gu, Julia Memar, and Yakov Zinder. Search strategies for problems with detectable boundaries and restricted level sets. In *Data and Decision Sciences in Action*, pages 149–162. Springer, 2018.

[41] Hanyu Gu, Julia Memar, and Yakov Zinder. Improved lagrangian relaxation-based optimisation procedure for scheduling with storage. *IFAC-PapersOnLine*, 52(13):100–105, 2019.

[42] Monique Guignard. Lagrangean relaxation. *Top*, 11(2):151–200, 2003.

[43] Monique Guignard. *Lagrangian Relaxation*. Springer US, 2013.

[44] Jatinder ND. Gupta. A functional heuristic algorithm for the flowshop scheduling problem. *Journal of the Operational Research Society*, 22(1):39–47, 1971.

[45] Dan Gusfield. Bounds for naive multiple machine scheduling with release times and deadlines. *Journal of Algorithms*, 5(1):1–6, 1984.

[46] Leslie A. Hall and David B. Shmoys. Approximation schemes for constrained scheduling problems. In *30th Annual Symposium on Foundations of Computer Science*, pages 134–139. IEEE, 1989.

[47] Nicholas G. Hall and Chelliah Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations research*, 44(3):510–525, 1996.

[48] Claire Hanen and Yakov Zinder. The worst-case analysis of the garey–johnson algorithm. *Journal of Scheduling*, 12(4):389–400, 2009.

[49] BR Heap. Permutations by interchanges. *The Computer Journal*, 6(3):293–298, 1963.

[50] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[51] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical programming*, 1(1):6–25, 1971.

[52] Michael Held, Philip Wolfe, and Harlan P. Crowder. Validation of subgradient optimisation. *Mathematical programming*, 6(1):62–88, 1974.

[53] Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1):144–162, 1987.

[54] Johannes A. Hoogeveen, Steef L. van de Velde, and Bart Veltman. Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Applied Mathematics*, 55(3):259–272, 1994.

[55] Te C. Hu. Parallel sequencing and assembly line problems. *Operations research*, 9(6):841–848, 1961.

[56] Takashi Irohara. Lagrangian relaxation algorithms for hybrid flow-shop scheduling problems with limited buffers. *International Journal of Biomedical Soft Computing and Human Sciences*, 15(1):21–28, 2010.

[57] Selmer M. Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1(1):61–68, 1954.

[58] Marc T. Kaufman. An almost-optimal algorithm for the assembly line scheduling problem. *IEEE Transactions on Computers*, 100(11):1169–1174, 1974.

[59] Alexander Kononov, Jen-Shin Hong, Polina Kononova, and Feng-Cheng Lin. Quantity-based buffer-constrained two-machine flowshop problem: active and passive prefetch models for multimedia applications. *Journal of Scheduling*, 15(4):487–497, 2012.

[60] Alexander Kononov, Polina Kononova, and Jen-Shin Hong. Two-stage multimedia scheduling problem with an active prefetch model. *IFAC Proceedings Volumes*, 42(4):2001–2006, 2009.

[61] Alexander Kononov, Julia Memar, and Yakov Zinder. Flow shop with job–dependent buffer requirements—a polynomial–time algorithm and efficient heuristics. In *International Conference on Mathematical Optimisation Theory and Operations Research*, pages 342–357. Springer, 2019.

[62] Polina Kononova and Yuri Kochetov. The variable neighborhood search for the two machine flow shop problem with a passive prefetch. *Journal of Applied and Industrial Mathematics*, 7(1):54–67, 2013.

[63] Manfred Kunde. Nonpreemptive lp-scheduling on homogeneous multiprocessor systems. *SIAM Journal on Computing*, 10(1):151–173, 1981.

[64] Shui Lam and Ravi Sethi. Worst case analysis of two scheduling algorithms. *SIAM Journal on Computing*, 6(3):518–536, 1977.

[65] Ailsa H. Land and Alison G. Doig. An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008*, pages 105–132. Springer, 2010.

[66] Eugene L. Lawler, Jan K. Lenstra, Alexander HG. Rinnooy Kan, and David B. Shmoys. Sequencing and scheduling: Algorithms and complexity. *Handbooks in operations research and management science*, 4:445–522, 1993.

[67] Eugene L. Lawler and David E. Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

[68] Hoang Thanh Le, Philine Geser, and Martin Middendorf. An iterated local search algorithm for the two-machine flow shop problem with buffers and constant processing times on one machine. In *European Conference on Evolutionary Computation in Combinatorial Optimisation (Part of EvoStar)*, pages 50–65. Springer, 2019.

[69] Rainer Leisten. Flowshop sequencing problems with limited buffer storage. *The International Journal of Production Research*, 28(11):2085–2100, 1990.

[70] Jan K. Lenstra, Alexander HG. Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. In *Annals of discrete mathematics*, volume 1, pages 343–362. Elsevier, 1977.

[71] Jan K. Lenstra and Alexander HR Rinnoy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, 1978.

[72] Joseph YT. Leung. *Handbook of scheduling: algorithms, models, and performance analysis*. CRC press, 2004.

[73] Feng-Cheng Lin, Jen-Shin Hong, and Bertrand MT. Lin. A two-machine flowshop problem with processing time-dependent buffer constraints—an application in multimedia presentations. *Computers & Operations Research*, 36(4):1158–1175, 2009.

[74] Feng-Cheng Lin, Jen-Shin Hong, and Bertrand MT. Lin. Sequence optimisation for media objects with due date constraints in multimedia presentations from digital libraries. *Information Systems*, 38(1):82–96, 2013.

[75] Feng-Cheng Lin, Chien-Yin Lai, and Jen-Shin Hong. Minimize presentation lag by sequencing media objects for auto-assembled presentations from digital libraries. *Data & Knowledge Engineering*, 66(3):382–401, 2008.

[76] Shi Qiang Liu and Erhan Kozan. Parallel-identical-machine job-shop scheduling with different stage-dependent buffering requirements. *Computers & Operations Research*, 74:31–41, 2016.

[77] Peter B. Luh and Debra J. Hoitomt. Scheduling of manufacturing systems using the lagrangian relaxation technique. *IFAC Proceedings Volumes*, 24(14):1–6, 1991.

[78] Peter B. Luh and Debra J. Hoitomt. Scheduling of manufacturing systems using the lagrangian relaxation technique. *IEEE Transactions on automatic control*, 38(7):1066–1079, 1993.

[79] Kun Mao, Quan-ke Pan, Xinfu Pang, and Tianyou Chai. A novel lagrangian relaxation approach for a hybrid flowshop scheduling problem in the steelmaking-continuous casting process. *European Journal of Operational Research*, 236(1):51–60, 2014.

[80] Julia Memar, Gaurav Singh, and Yakov Zinder. Scheduling partially ordered uet tasks on dedicated machines. *IFAC Proceedings Volumes*, 46(9):1672–1677, 2013.

[81] Julia Memar, Yakov Zinder, and Alexander Kononov. Worst-case analysis of a modification of the brucker-garey-johnson algorithm. In *International Conference on Optimisation Problems and Their Applications*, pages 78–92. Springer, 2018.

[82] Hugo Hissashi Miyata and Marcelo Seido Nagano. The blocking flow shop scheduling problem: A comprehensive and conceptual review. *Expert Systems with Applications*, 137:130–156, 2019.

[83] David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe, and Edward C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.

[84] Muhammad Nawaz, E. Emory Enscore Jr, and Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

[85] Douglas S. Palmer. Sequencing jobs through a multi-stage process in the minimum total time—a quick method of obtaining a near optimum. *Journal of the Operational Research Society*, 16(1):101–107, 1965.

[86] Christos H. Papadimitriou and Paris C. Kanellakis. Flowshop scheduling with limited temporary storage. *Journal of the ACM (JACM)*, 27(3):533–549, 1980.

[87] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2016.

[88] Chris N. Potts and Vitaly A. Strusevich. Fifty years of scheduling: a survey of milestones. *Journal of the Operational Research Society*, 60(1):S41–S68, 2009.

[89] Hans Röck. The three-machine no-wait flow shop is np-complete. *Journal of the ACM (JACM)*, 31(2):336–345, 1984.

[90] Rubén Ruiz and Concepción Maroto. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494, 2005.

[91] Rubén Ruiz, Eva Vallada, and Carlos Fernández-Martínez. Scheduling in flowshops with no-idle machines. In *Computational intelligence in flow shop and job shop scheduling*, pages 21–51. Springer, 2009.

[92] Robert Sedgewick. Permutation generation methods. *ACM Computing Surveys (CSUR)*, 9(2):137–164, 1977.

[93] Robert Segewick. Permutation generation methods. In *Talk at Dagstuhl Workshop on Data Structures, Wadern, Germany*, 2002.

[94] Gaurav Singh and Yakov Zinder. Worst-case performance of two critical path type algorithms. *Asia-Pacific Journal of Operational Research*, 17(1):101, 2000.

[95] Ola Svensson. Conditional hardness of precedence constrained scheduling on identical machines. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 745–754. ACM, 2010.

[96] Eric Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European journal of Operational research*, 47(1):65–74, 1990.

[97] Eric Taillard. Benchmarks for basic scheduling problems. *European journal of operational research*, 64(2):278–285, 1993.

[98] Li-Xin Tang and Hua Xuan. Lagrangian relaxation algorithms for real-time hybrid flowshop scheduling with finite intermediate buffers. *Journal of the Operational Research Society*, 57(3):316–324, 2006.

[99] Lixin Tang, Peter B Luh, Jiyin Liu, and Lei Fang. Steel-making process scheduling using lagrangian relaxation. *International Journal of Production Research*, 40(1):55–70, 2002.

[100] Lixin Tang, Hua Xuan, and Jiyin Liu. A new lagrangian relaxation algorithm for hybrid flowshop scheduling to minimize total weighted completion time. *Computers & Operations Research*, 33(11):3344–3359, 2006.

[101] Jeffrey D. Ullman. Np-complete scheduling problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975.

[102] Steven L. van de Velde. Machine scheduling and lagrangian relaxation. 1991.

[103] Andreas Witt and Stefan Voß. Simple heuristics for scheduling with limited intermediate storage. *Computers & Operations Research*, 34(8):2293–2309, 2007.

[104] Gerhard J. Woeginger. Heuristics for parallel machine scheduling with delivery times. *Acta Informatica*, 31(6):503–512, 1994.

[105] Minyi Yue. On the exact upper bound for the multifit processor scheduling algorithm. *Annals of Operations Research*, 24(1):233–259, 1990.

[106] Yakov Zinder. An iterative algorithm for scheduling UET tasks with due dates and release times. *European Journal of Operational Research*, 149(2):404–416, 2003.

[107] Yakov Zinder. The strength of priority algorithms. *Proceedings, MISTA*, pages 531–537, 2007.

[108] Yakov Zinder, Alexander Kononov, and Joey Fung. A 5-parameter complexity classification of the two-stage flow shop scheduling problem with job dependent storage requirements. *Journal of Combinatorial Optimization*, pages 1–34, 2021.

[109] Yakov Zinder, Julia Memar, and Gaurav Singh. Discrete optimisation with polynomially detectable boundaries and restricted level sets. *Journal of Combinatorial Optimisation*, 25(2):308–325, 2013.

[110] Yakov Zinder and Duncan Roper. A minimax combinatorial optimisation problem on an acyclic directed graph: polynomial-time algorithms and complexity. In *Proceedings of the AC Aitken centenary conference, Dunedin*, pages 391–400, 1995.

[111] Yakov Zinder and Duncan Roper. An iterative algorithm for scheduling unit-time taskswith precedence constraints to minimisethe maximum lateness. *Annals of Operations Research*, 81:321–343, 1998.

[112] Yakov Zinder, Gaurav Singh, and Rene Weiskircher. A new method of scheduling UET tasks on parallel machines. *International MultiConference of Engineers & Computer Scientists 2006*, pages 796 – 801, 2006.

[113] Yakov Zinder, Bo Su, Gaurav Singh, and Ron Sorli. Scheduling UET-UCT tasks: branch-and-bound search in the priority space. *Optimisation and Engineering*, 11(4):627–646, 2010.